НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«До захисту допущено»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

спеціальності 121 Інженерія програмного забезпечення

на тему: «Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS»

Виконав:
студент IV курсу, групи КП-61
Казимиров Данило Миколайович _____

Керівник:
Доцент кафедри ПЗКС, к.т.н., доцент,
Заболотня Тетяна Миколаївна _____

Консультант з нормоконтролю:
Доцент кафедри ПЗКС, к.т.н., доцент,
Онай Микола Володимирович _____

Рецензент:
Доцент кафедри ММСА ІПСА, к.т.н., доцент,
Дідковська Марина Віталіївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет прикладної математики
Кафедра програмного забезпечення комп'ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення комп'ютерних та інформаційно-пошукових систем»

ЗАТВЕРДЖУЮ
Науковий керівник кафедри
_____ Іван ДИЧКА
«___»_____2019 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Казимирову Данилу Миколайовичу

1. Тема проєкту «Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS», керівник проєкту Заболотня Тетяна Миколаївна, доцент кафедри ПЗКС, к.т.н., доцент, затверджені наказом по університету від «25» травня 2020 р. №1181-с

2. Термін подання студентом проєкту «12» червня 2020 р.

3. Вихідні дані до проєкту: див. Технічне завдання.

4. Зміст пояснювальної записки:
   - обґрунтування вибору засобів реалізації;
   - структурно-алгоритмічна організація;
   - особливості реалізації програмного забезпечення;

5. Перелік обов'язкового графічного матеріалу:
   - алгоритм створення чату особою, яка знайшла чужу власність (креслення);
   - структура бази даних (креслення);
   - архітектура веб-додатку (плакат);
   - дерево проблем (плакат).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Онай М.В., доцент		

7. Дата видачі завдання «31» жовтня 2019 р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	Вивчення літератури за тематикою проєкту	14.11.2019	
2.	Розроблення та узгодження технічного завдання	28.11.2019	
3.	Розроблення структури веб-додатку	15.12.2019	
4.	Підготовка матеріалів першого розділу дипломного проєкту	30.12.2019	
5.	Розроблення дизайну сторінок та графічних елементів	03.02.2020	
6.	Підготовка матеріалів другого розділу дипломного проєкту	20.02.2020	
7.	Програмна реалізація веб-додатку	10.03.2020	
8.	Тестування web-ресурсу	17.03.2020	
9.	Підготовка матеріалів третього розділу дипломного проєкту	30.03.2020	
10.	Підготовка матеріалів четвертого розділу дипломного проєкту	11.04.2020	
11.	Підготовка графічної частини дипломного проєкту	21.04.2020	
12.	Оформлення документації дипломного проєкту	26.05.2020	

Студент Данило КАЗИМИРОВ

Керівник проєкту Тетяна ЗАБОЛОТНЯ

АНОТАЦІЯ

Даний дипломний проект присвячений створенню веб-додатку для сприяння поверненню втрачених речей на основі платформи AWS.

Веб-додаток реалізовано у вигляді веб-сайту, призначеного для користування особами, які бажають запобігти втраті своєї власності шляхом персоналізації речей через нанесення на них зображень із QR-кодом, який ідентифікує їх господаря. При скануванні зображення із QR-кодом, людина потрапить на сторінку чату із його власником. Таким чином створюється початковий прецедент процесу повернення речі її господарю. Метою створення даного проєкту є підвищення відсотку повернутих загублених речей. Розглянуто сучасні технології та підходи до розроблення програмного забезпечення, обґрунтовано доцільність проектування сервіс-орієнтованої архітектури веб-додатків за моделлю без серверних обчислень на основі платформи хмарних обчислень AWS.

У даному дипломному проекті розроблено: архітектуру веб-додатку побудовану за сервіс-орієнтованою моделлю, алгоритм авторизації користувача та підтвердження правдивості його намірів щодо повернення чужої власності, процедуру завантаження зображень у хмарне сховище, а також графічні елементи та дизайн веб-сторінок.

ABSTRACT

This diploma project deals with the development of a web-based application to facilitate the return of lost things based on the AWS platform.

The web application is implemented as a website intended for use by individuals who wish to prevent the loss of their property by personalizing their things through the application of images on them with a QR code that identifies their owner. When scanning an image with the QR code, a person will be taken to a chat page with its owner. This creates an initial precedent for the process of returning the item to its owner. This project aims to increase the percentage of returned things that have been lost. Modern technologies and approaches to software development are considered, the feasibility of designing a service-oriented architecture of web applications using a serverless model based on the AWS cloud computing platform is justified.

While working on this project, the following artefacts have been developed: the architecture of the web-application based on the service-oriented model, algorithm of authorization of the user and acknowledgment of truthfulness of its intentions on the return of another's property, the procedure of image loading in cloud storage, and also graphic elements and design of web pages.

АННОТАЦИЯ

Данный дипломный проект посвящен созданию веб-приложения для содействия возвращению утраченных вещей на основе платформы AWS.

Веб-приложение реализовано в виде веб-сайта, предназначенного для использования лицами, желающими предотвратить потерю своей собственности путем персонализации вещей через нанесение на них изображений с QR-кодом, который идентифицирует их хозяина. При сканировании изображения с QR-кодом, человек попадет на страницу чата с его владельцем. Таким образом создается начальный прецедент процесса возврата вещи ее хозяину. Целью создания данного проекта является повышение процента возвращенных утерянных вещей. Рассмотрены современные технологии и подходы к разработке программного обеспечения, обоснована целесообразность проектирования сервис-ориентированной архитектуры веб-приложений с использованием модели без серверных вычислений на основе платформы облачных вычислений AWS.

В данном дипломном проекте разработано: архитектуру веб-приложения основаную на сервис-ориентированной модели, алгоритм авторизации пользователя и подтверждения правдивости его намерений по возврату чужой собственности, процедуру загрузки изображений в облачное хранилище, а также графические элементы и дизайн веб-страниц.

ДП.045440-01-90. Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS. Відомість проєкту

Позначення	Найменування	Кіл-ть	Примітка
	Документація проєкту		
ДП.045440-02-91	Веб-додаток для сприяння	5	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Технічне завдання		
ДП.045440-03-81	Веб-додаток для сприяння	81	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Пояснювальна записка		
ДП.045440-04-51	Веб-додаток для сприяння	4	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Програма та методика		
	тестування		
ДП.045440-05-34	Веб-додаток для сприяння	10	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Керівництво користувача		

Позначення	Найменування	Кіл-ть	Примітка
ДП.045440-06-99	Веб-додаток для сприяння	1	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Структура бази даних.		
	ERD діаграма		
ДП.045440-07-99	Веб-додаток для сприяння	1	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Взаємодія особи, яка		
	знайшла чужу річ, із веб-		
	додатком. Діаграма		
	діяльності		
ДП.045440-08-98	Веб-додаток для сприяння	1	
	поверненню втрачених		
	речей на основі		
	платформи AWS.		
	Компакт-диск		

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS

Технічне завдання

ДП.045440-02-91

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Данило КАЗИМИРОВ

2019

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS.

Галузь застосування: інформаційні технології.

2. ПІДСТАВА ДЛЯ РОЗРОБЛЕННЯ

Підставою для розроблення є завдання на дипломне проєктування, затверджене кафедрою програмного забезпечення комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» (КПІ ім. Ігоря Сікорського).

3. ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для використання особами, які бажають запобігти втраті своєї власності шляхом персоналізації речей через нанесення на них зображень із QR-кодом, який ідентифікує їх господаря. В QR-коді закодовано посилання на сторінку даного веб-додатку із чатом з господарем втраченої речі.

4. ВИМОГИ ДО ПРОГРАМНОГО ПРОДУКТУ

Веб-додаток повинен забезпечувати такі основні функції:

1) можливість динамічного оновлення вмісту веб-сторінок;
2) можливість вести чат між двома користувачами;
3) розгортка в оточенні платформи хмарних обчислень AWS;
4) можливість створювати та завантажувати QR-коди авторизованими користувачами;
5) Можливість перейти на сторінку чату за посиланням в QR-коді;

Розробку виконати на платформі Node.js з використанням мови програмування TypeScript з використанням технології React.js та AJAX.

Додаткові вимоги:

1) відправка оповіщень на електрону пошту користувача;
2) адаптивний користувацький інтерфейс;
3) використання статичного доменного імені;
4) реалізація в інтерфейсі прив’язки кожного чату до відповідного QR-коду;

5. ВИМОГИ ДО ПРОЄКТНОЇ ДОКУМЕНТАЦІЇ

У процесі виконання проєкту повинна бути розроблена наступна документація:

1) пояснювальна записка;
2) програма та методика тестування;
3) керівництво користувача;
4) креслення:
   - «Алгоритм дій ОЯЗЧВ для початку спілкування із ВЗР. Схема роботи програмної системи»;
   - «Структура бази даних. ERD-діаграма».

6. ЕТАПИ ПРОЄКТУВАННЯ

Вивчення літератури за тематикою роботи……..14.11.2019

Розроблення та узгодження технічного завдання……....28.11.2019

Розроблення структури веб-додатку…….....15.12.2019

Розроблення дизайну сторінок та графічних елементів……..03.02.2020

Програмна реалізація веб-додатку……...17.03.2020

Тестування веб-додатку…….....03.04.2020

Підготовка матеріалів текстової частини проєкту……..….28.04.2020

Підготовка матеріалів графічної частини проєкту……...…12.05.2020

Оформлення технічної документації проєкту……..…25.05.2020

7. ПОРЯДОК ТЕСТУВАННЯ РОЗРОБКИ

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»
Науковий керівник кафедри
_____ Іван ДИЧКА
«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS

Пояснювальна записка

ДП.045440-03-81

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Данило КАЗИМИРОВ

2020

ЗМІСТ

СПИСОК ТЕРМІНІВ, СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення;

Веб-браузер – програмне забезпечення, що дає можливість взаємодіяти з елементами на гіпертекстовій веб-сторінці;

iOS – ОС компанії Apple Inc. для їх мобільних телефонів iPhone;

ЦП – Центральний процесор;

ОС – операційна система;

СУБД – система управління базами даних;

БД – база даних;

JWT – Json Web Token – відкритий стандарт створення токенів доступа, на основі формату представлення даних JSON;

Транзакція – група атомарних послідновних операцій із БД, може бути виконана або цілком і успішно, або ніяк;

ОЯЗЧВ – особа яка знайшла чужу власність;

ЗБ – загублена річ;

ВЗР – власник загубленої речі;

QR-код – двомірний штрих-код, розроблений для представлення інформації у вигляді зображення, яке зчитується та декодується апаратними засобами;

Штрих-код – спосіб представлення даних у форматі, зручному для зчитування машиною;

SQL Injection – вид атаки на веб-сайти, який полягає у впровадження в запит довільного SQL-коду, який буде оброблений веб-сервером;

XSS – тип вразливості інформаційних систем у мережі Інтернет, який полягає у впровадженні користувацьких скриптів у сторінки, віддані веб-сервером;

CSRF – тип веб-атаки, що полягає у виконанні певних дій від імені користувача на веб-сторінці де останній авторизований;

SQL – структурована мова запитів до БД

ACID – набір властивостей, які гарантують надійність виконання

транзакцій у БД;

OAuth2 – відкритий стандарт авторизації користувачів шляхом надання інформації користувача, яка зберігається одним ресурсом іншому.

Push-нотифікація – один із методів поширення інформації, коли дані потрапляють від відправника до одержувача на основі встановлених параметрів;

drag'n'drop – спосіб керування інтерфейсом користувача за допомогою перетаскування графічних елементів;

AWS – Amazon Web Services, комерційна платформа хмарних обчислень яка розроблена і підтримується компанією Amazon;

AWS Lambda – послуга платформи AWS, яка забезпечує подійно-орієнтовані без серверні обчислення;

AWS SQS – послуга платформи AWS, яка полягає в надані черги повідомлень для користування в рамках платформи AWS;

AWS SES – послуга платформи AWS, яка полягає в надані сервісу відправки та отримання повідомлень через електронну пошту;

CDN – географічно розподілена мережева інфраструктура, яка дозволяє оптимізувати доставка і розповсюдження контенту кінцевим користувачам;

GCP – Google Cloud Platform комерційна платформа хмарних обчислень яка розроблена і підтримується компанією Google LLC.

ВСТУП

Пересічна людина двадцять першого століття зазвичай володіє наступними речами: мобільний телефон, персональний комп'ютер, записник, фотокамера, гаманець, навушники, документи тощо. Окрім матеріальної цінності, ці речі несуть в собі цінність інформації, яку зберігають, або спогади їх власника, пов'язані із собою. Беручи до уваги, що на загубленому запам'ятовуючому пристрої може знаходитися інформація, що стосується професійної діяльності людини, збиток спричинений його втратою може перевищувати цінність самого пристрою в десятки та сотні разів.

З поміж людей ніхто не застрахованим від втрати свого майна. Це може бути крадіжка, або прикрий перебіг обставин, за яких річ була загублена під час подорожі за кордоном або навіть вранці в метро по дорозі на роботу. Звісно, що кожен, хто стикався з даною ситуацією, хоче повернути свою власність. В цей момент перед ним постає нетривіальне запитання, як це зробити.

Зазвичай сумлінні люди, які знайшли чужу річ, шукають інформацію про її власника або на самій речі або у засобах масової інформації з ціллю повернути її господарю. В той самий час той, хто її загубив, намагається розповсюдити інформацію про себе та свою проблему. Це не ефективно та створює додаткові вразливості, якими користуються шахраї. Вони можуть скористатися ситуацією з ціллю заволодіння річчю або грошима, які власник може заплатити в якості винагороди за її повернення.

Метою даного дипломного проєкту є збільшення відсотку повернутих речей шляхом розроблення системи пошуку власника загубленої речі за QR-кодом на ній, який містить посилання на веб-сторінку, з якої можна безпечно для обох сторін зв'язатися із власником речі та домовитися про повернення останньої. Це дозволить вирішити проблему ідентифікації господаря речі та встановлення контакту між ним і людиною, яка знайшла його власність, захищаючи обох від дій шахраїв. Тому розроблення веб-

додатку для сприяння поверненню втрачених речей є актуальним на сьогоднішній день.

1. ОГЛЯД ІСНУЮЧИХ ПРОГРАМНИХ РІШЕНЬ

1.1. Огляд проблеми, яка вирішується ПЗ

Щохвилини люди втрачають свою власність. Відомо [1], що кожного тижня тільки в аеропортах США губляться 12000 ноутбуків. Головне питання, перед яким постає людина, коли втрачає власність – як мені її повернути. Також існує кілько додаткових питань: як зробити це швидко, надійно та витративши мінімум ресурсів і не псуючи собі нерви.

Відповісти пересічній людині на ці питання важко. Як правило, на більшість речей люди не додають свою персональну інформацію, через оману, що вони застраховані від крадіжки або хвилюючись за збереження конфіденційності. В такому разі ідентифікувати власника пропажі майже неможливо. За даними Uber [2] до списку речей, які найчастіше втрачають користувачі таксі, відносяться: ключі, окуляри, навушники та фотоапарати. Ці речі не містять інформацію про свого господаря, що викликає труднощі із його ідентифікацією, через що такі речі й не повертають їх власникам.

Однією з перших ідей, які спадають на думку людині, яка знайшла чужу власність є віднести її до відділу поліції. В Токіо 80% загублених речей повертаються до своїх власників [3]. Такі показники стали можливими завдяки високій щільності на квадратний кілометр малих відділень поліції та відповідальності і порядності людей східно культури. Для порівняння, в місті Нью-Йорк цей показник менше 10%.

Інший вихід із ситуації – написати на публічному ресурсі про знахідку та чекати, поки той, хто її загубив, побачить замітку. Однак існує великий ризик стати жертвою шахраїв, які вдають із себе власника пропажі з метою її заволодіння.

З іншого боку власник речі намагатиметься повідомити про себе якомога більшому загалу. На думку пересічного громадянина кращим засобом зробити це є повідомлення в ЗМІ. Результат цих дій зазвичай є діаметрально протилежним очікуваному: заміст повернення власності, людина втрачає ще й гроші, які заплатить як винагороду шахраям, які

пообіцяють повернути її після отримання коштів а потім зникають.

Звісно існує перелік речей, котрі несуть в собі інформацію про свого господаря, за якою можна зв’язатися із ним. До їх переліку можна віднести паспорт, водійське посвідчення, гаманець, шкільний зошит. За ПІБ людини на цих речах легко можна знайти її у соціальних мережах. Проте знову ж таки постає питання у тому, як не помилитися та не написати повідомлення шахраєві, який може видавати себе за іншу людину.

Тому основною проблемою, яка вирішується розробленим ПЗ є відсутність надійного засобу передати інформацію про власника речі на його особистих речах за допомогою якої із ним можна зв’язатися.

1.2. Опис вимог до розроблюваного ПЗ

На початку роботи над ПЗ в рамках цього дипломного проєкту було сформовано основні вимоги до нього. По-перше, система має зберігати дані про своїх користувачів та надавати кожному з них можливість завантажувати на свій ПК зображення QR-коду [4], в який буде закодовано посилання на веб-сторінку із його формою для створення повідомлення для її власника. По-друге, форма на яку переходить користувач, коли відскановує мітку на знайденому ним предметі, має вимагати від нього завантаження фото із зображенням фото речі, на якій він знайшов QR-код. По-третє, якщо власник погоджується розпочати спілкування, сервіс має надавати цю можливість. Користувачі мають мати змогу анонімно обмінюватися повідомленнями, доки вони не вирішать відкрити свої контакти один одному. Оскільки зазвичай QR-коди сканують за допомогою мобільних пристроїв, веб-інтерфейс має бути зручним для взаємодії із ним за допомогою планшетів та телефонів.

Таким чином можна узагальнити приведені вище вимоги:

- ідентифікація власника предмету за QR-кодом;
- можливість вести бесіду між тим, хто знайшов чужу річ, і тим, хто її загубив;

- веб інтерфейс.

Отже, після розгляду зазначених вимог, було проведено детальний порівняльний аналіз існуючих програмних рішень. Також враховано їхні переваги та недоліки при розробленні програмного застосунку.

1.3. Аналіз існуючих рішень

1.3.1. *Find My*

Find My – це програмне забезпечення, яке поставляється у складі пакету послуг та сервісів від компанії Apple Inc [5]. Дія цього ПЗ розповсюджується на пристрої, розроблені цією компанією із версією ОС, яке відповідає вимогам. Сервіс був є інтегрованим в iOs, watchOs та macOS. Починаючи із iOs 13 та macOs Cataline це ПЗ включено у склад цих ОС.

До основних функцій цього рішення відносяться:

- пошук гаджетів та їх власників за їх геопозицією;
- можливість програвати музику, якщо пристрій увімкнено;
- можливість перевести прилад у режим “вкрадений”, що вимагатиме введенню паролю для подальшої роботи;
- можливість віддалено назавжди заблокувати пристрій, тим самим позбавивши злодія можливості його продати;
- виведення повідомлення від власника на екрані пристрою.

Недоліком цієї системи є те, що для своєї роботи це ПЗ потребує постійного підключення до мережі Інтернет, водночас суттєвою перевагою Find My у порівнянні із аналогами є можливість пристроїв останніх моделей передавати інформацію про своє положення навіть коли вони вимкнені користуючись підключенням до мережі через сусідні пристрої на яких встановлене це ПЗ, а воно може бути встановлено лише на пристрої виробництва Apple.

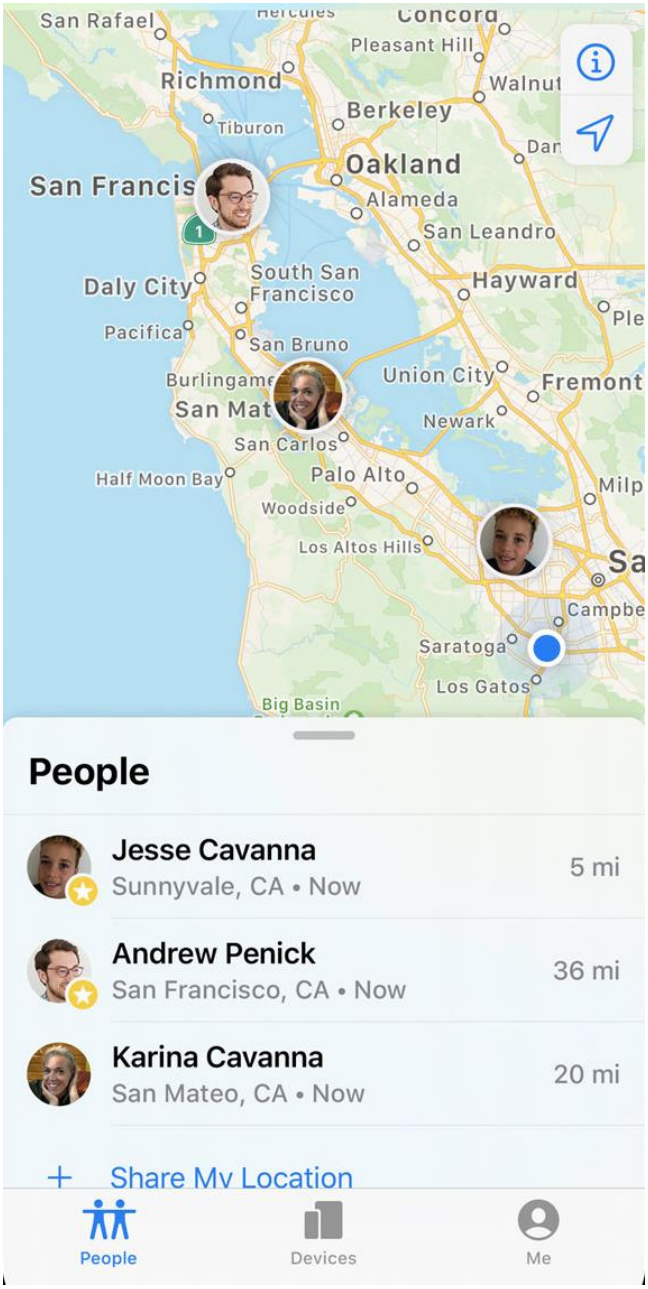


Рис. 1.1. Перегляд інформації про геопозицію користувачів за позицією їх мобільних телефонів

Принцип роботи Find My полягає у постійній синхронізації між пристроєм та сереверами компанії Apple. На них вони відправляють дані про своє місцезнаходження. Завдяки цьому їх власник має змогу в режимі реального часу отримувати інформацію про їх геопозицію на веб-сайті додатку iCloud. Якщо власник не має доступу до свого акаунту iCloud він не зможе скористатися Find My.

Рис. 1.2. Вивід повідомлення від власника на дисплеї Apple Watch

Іншим суттєвим недоліком є те, що повідомлення можуть бути відправлені лише в одному боці. Той, хто зараз володіє пристроєм, не зможе зв'язатися з його допомогою з його власником, тому зазвичай у це повідомлення люди додають свою контактну інформацію.

Підсумовуючи вищесказане, можна виділити основні недоліки цього ПЗ:

1. Жорстка прив'язаність до пристроїв виробництва Apple Inc.
2. Потреба у останніх версіях ОС.
3. Потреба у постійному підключенні до мережі Інтернет.
4. Прив'язаність до акаунту iCloud.
5. Обмежений функціонал обміну повідомленнями.

1.3.2. *Пошук загубленого багажу в аеропортах*

Завжди є ризик втратити свою власність у місцях скупчення великої кількості людей. Одним з таких місць є аеропорти та залізничні вокзали. Проте переважно лише в аеропортах використовується маркування речей пасажирів. Це викликано тим, що пасажири зобов'язані здати свій багаж. Беручи до уваги великий пасажиропотік аеропортів, шанс їх співробітників загубити чиюсь поклажу дуже високий. Тому використання міток із баркодом на багажі в цих установах є доцільним та вирішує проблему

ідентифікації власника.

Рис. 1.3. Наліпка на багаж у від авіакомпанії “SAS” [6]

Алгоритм дій пасажира при втраті своїх речей наступний.

1. Звернутися до співробітників аеропорту.
2. Надати їм свої документи, що посвідчують особу, та квиток.
3. Дочекатися, доки працівники відшукають у базі знайдених речей власність пасажира за міткою на ній, або залишити свої контакти, за якими співробітники аеропорту зможуть зв’язатися з ним.
4. Пересвідчитися, що всі речі на місці.

Перевагами цього підходу є:

- Централізований пошук багажу.
- Точна ідентифікація власника речі за його посадковим талоном.
- Єдина система пошуку багажу, яка використовується у всіх аеропортах світу.

- Повідомлення пасажира про його знайдену власність навіть після того, як він залишив будівлю аеропорту.
- Відшкодування вартості втраченого багажу, якщо він бух застрахований на момент покупки білету на літак.

В той самий час недоліками є:

- Може використовується лише користувачами послуг аеропорту.
- Застосовується лише до речей, які здаються в багажне відділення.
- Особисті речі, які пасажир бере із собою в салон літака не маркіруються і, як результат, не відслідковуються
- Речі будуть знайдені лише за умови, що вони не були вкрадені або загублені за межами аеропорту.

1.3.3. *Оберіг для важливого*

“Оберіг для важливого” – це пакет послуг від ВАТ “Паритет банк” [7], яка полягає в підтримці їх клієнтів і поверненню втрачених речей їх власникам, а також послуги організації страхових виплат в тих випадках, коли втрачений предмет не був знайденим. Сервіс забезпечує швидке і безпечне повернення законному власнику таких цінних речей, як ключи, документи та гаманці. Клієнт цього сервісу має придбати пакет послуг, який, в залежності від обраної конфігурації, може містити декілька брелків-міток, жетонів або наліпок із зображенням мітки. Їх він має розмістити на своїх речах, які є дуже коштовними для нього.

“Оберіг для важливого” зберігає в своїй базі даних інформацію про своїх клієнтів та ідентифікатор кожної з міток, яку він придбав. Коли людина загубить річ із міткою на ній, той, хто її знайде, повинен звернутися до операторів сервісу та повідомити про свою знахідку. Оператор буде намагатися домовитися про її передачу співробітникам сервісу. Якщо спроба була вдалою, кур’єр доставить власнику його пропажу.

До переваг цього сервісу можна віднести:

- фізичні брелки і наліпки із мітками високої якості;

- відшкодування збитків в разі втрати власності;
- доставка кур’єром знайдених предметів клієнту;
- зацікавленість банку у знайдені консенсусу із тим, хто знайшов чужу власність, через необхідність виплати клієнту страхового відшкодування;
- кожна мітка містить інформацію, про те, що в разі повернення цього предмета власнику, той, то це зробить, отримає винагороду. Це мотивує людей звертатися до операторів;
- відповідальність за процес переговорів із тим, хто знайшов чужу річ бере на себе компанія.

Рис. 1.4. Предмети, які входять до складу преміум пакету послуг “Оберіг для важливого”

Очевидним недоліком такої системи є те, що користувач має придбати пакет послуг із брелками або наліпками. Для цього йому слід дочекатися на доставку поштою своєї покупки або особисто забрати їх у представника компанії. Компанія не дозволяє клієнтам власноруч роздруковувати наліпки

та створювати брелки.

Також існує ймовірність, що необхідність встановлювати особистий контакт із оператором, може мати негативний вплив на кількість телефонних звернень до оператору.

До недоліків бізнес-моделі цього сервісу відноситься необхідність платити не за кінцевий результат, а лише за шанс, що той, хто знайшов чужу власність, зв'яжеться із операторами "Паритет банку", а вони в свою чергу не пошкодують коштів і сил на те, аби повернути знахідку її власнику. Із очевидних причин сервіс це не афішує.

Підсумовуючи вищесказане можемо виділити наступні недоліки:

- той, хто знайшов чужу власність має зв'язуватися із оператором, тим самим відкриваючи йому свій номер телефона або адресу електронної пошти;
- користувачі платять не за фактичний результат, а за невідомий шанс, що їх власність буде повернена;
- висока вартість кожного пакету із мітками;
- для користуванням сервісом, людина має бути клієнтом "Паритет банку";
- необхідність кожного разу замовляти брелки та наліпки в банку, заборона створювати мітки власноруч.

1.4. Результати проведеного аналізу

Виходячи з отриманих даних після проведення аналізу, можна стверджувати, що описані рішення повністю не вирішують задачу сприяння поверненню загублених предметів їх власнику. Не існує універсального методу повернення людині втраченої нею власності. Всі існуючі рішення обмежені зоною можливого застосування (аеропорт, ресторан, тощо), переліком речей, на які розповсюджується це рішення (телефон, часи, тощо), потребують укладення договору про надання послуг та плати за нього або обмежують у використанні створених власноруч міток.

Таким чином, для систем які допомагають поверненню втрачених речей було виділено ряд обов'язкових функцій:

- надання можливості користувачу самому зв'язуватися із тим, хто знайшов його речі;
- надання можливості користувачу самому створювати трекери для його речей;
- відсутність обмежень на використання системи в рамках однієї локації;
- страхові компенсації в разі невдалої спроби повернення речей;
- обмежений перелік речей, на які розповсюджується дія сервісу;
- отже, сформовано порівняльну таблицю результатів аналізу програмних рішень (табл. 1.1).

Таблиця 1.1

Результати аналізу існуючих рішень

Критерії / Рішення	Можливість користувачу зв' язуватися із тим, хто знайшов його власність	Необмежена кількість безкоштовних трекерів	Відсутність обмежень локації	Страхові компенсації	Обмежений перелік речей, на які розповсюджується дія сервісу	Захист від шахраїв
Find My	+	+/–	+	–	–	+
Пошук в аеропортах	+/–	–	+	+	–	+
Оберіг для важливого	–	–	–	+	+	+

Таким чином, проаналізовано рішення, розроблені відомими компаніями. На основі цих даних було описано функцілнальні вимоги до

сервісу, розробленого в рамках цього дипломного проєкту:

- можливість користувача власноруч створювати наліпки із QR-кодами, прив’язані до свого власника;
- можливість користувача редагувати сторінку, яку побачить людина, що перейшла за посиланням у QR-коді;
- веб-інтерфейс для швидкого доступу з будь-якого пристрою;
- вимога додати фото знайденого предмета до заявки на початок спілкування із ВЗР для підтвердження факту знайдення ОЯЗЧР його власності;
- реалізація захищеного чату між власником та людиною, яка перейшла за посиланням у QR-коді;
- можливість користувача спілкуватися із тим, хто знайшов його власність.

2. ОБҐРУНТУВАННЯ ВИБОРУ ЗАСОБІВ РЕАЛІЗАЦІЇ

Після описання функціональних вимог до веб-додатку, наступним кроком його реалізації є вибір засобів розробки. До них відносяться: мова програмування, відповідні фреймворки та бібліотеки до неї, СУБД, архітектурні шаблони. Основуючись на тому факті, що ПЗ має бути реалізовано у вигляді веб-додатку, доцільним є розділення функціональних модулів системи на серверну та клієнтську частини. Серверна частина відповідає за обробку запитів від клієнтської частини та керуватиме даними, які зберігаються в БД. Клієнтська частина є інтерфейсом, із яким взаємодіє користувач, та являє собою веб-сторінку, яку можна відкрити у мобільному та настільному браузері.

2.1. Вибір мови програмування для розроблення серверної частини

На основі описаних вище вимог, було сформовано перелік критеріїв, яким має відповідати мова програмування серверної частини веб-додатку. До цих критеріїв відносяться:

- наявність інструментів функціональної парадигми;
- наявність жорсткої типізації;
- асинхронна модель виконання коду;
- наявність фреймворку для побудови веб-серверу;
- використання парадигми ООП.

Використання засобів, які відповідають цим критеріям, дозволить зосередитися на розробленні ПЗ та пришвидшить процес прототипування веб-додатку для сприяння поверненню втрачених речей.

Під час вибору мови програмування було розглянуто мови, які є популярні та активно розвиваються на поточний момент. Далі наводяться результати аналізу кожної з них.

2.1.1. *Go*

Go був представлений у листопаді 2009 року працівниками Google. Метою розробки було створення виразного, високоефективного як при компіляції, так і при виконанні програм мови програмування, що дозволяє легко і просто писати надійні високоінтелектуальні програми. З моменту виходу Go отримав величезну популярність як мова програмування для написання веб-додатків та систем * -as-a-Service [8].

Ця мова має синтаксис дещо схожий на синтаксис мови C і C++, але привносить до них нові конструкції та шаблони проєктування, які не притамані іншим мовам [9]. роте із спрощеною моделлю керування пам’яттю за рахунок використання автоматичного прибиральника сміття. Go стала мовою, яка одразу розроблялася для використання під час розроблення серверів.

До переваг цієї мови можна віднести такі її характерні особливості:

- мова є компільованою: вихідний код перед виконанням має бути зібраний у бінарний виконуваний файл. За рахунок чого, написане нею ПЗ є одним із найшвидших у порівнянні із аналогічним ПЗ, написаним іншою мовою програмування. Для кожної ОС слід компілювати програму окремо;
- використовується строга статична типізація: перевірка типів відбувається у момент компіляції. За рахунок цього більшість помилок може бути усунена ще на моменті збірки програми;
- асинхронність досягається за рахунок багатопоточності, засобами горутин – більш легковісних аналогів потоків ОС;
- наявність багатої бібліотеки: товариство розробників вже створило широку кількість бібліотек, якими може бути вирішена більшість прикладних задач інженерів ПЗ;
- код компілятора мови є відкритим: мова розробляється ентузіастами. Її вихідний код доступний для перегляду всіма охочими. Таким чином із стандартної бібліотеки мови можна

підчерпнути знання особливостей побудови програм цією мовою;

- оптимізація для використання в системах із багатоядерними ЦП. За рахунок використання горутин, мова Go дозволяє оптимізувати ресурси ЦП та отримати максимум продуктивності.

До недоліків мови можна віднести:

- особливості роботи із ООП: автори Go пропагандують відмову від класів на користь використання структур і інтерфейсів. Мова не надає засобів до реалізації наслідування, пропонуючи використання композиції типів. При використанні мови Go не існує методів до створення узагальнених функцій та типів, відсутнє перевантаження методів;
- відсутність механізму обробки помилок: в Go не має синтаксичних конструкцій try-catch та throw, що притаманні іншим мовам програмування. В цій мові пропонується повертати інформацію про помилку в якості додаткового результату виконання функції та обробляти його в умовних конструкціях if;
- синтаксичні конструкції Go дещо відрізняються від інших мов, це стосується операцій із масивами, структурами та строками.

2.1.2. *Python*

Python – високорівнева інтерпретована, динамічна строго типізована мова програмування. Створена у 1991 році Гвідо ван Россумом. Актуальна версія мови Py3k вийшла у 2008 році. Застосовується у багатьох сферах програмування: розроблення веб-серверів, машинне (Machine learning) та глибоке навчання (Deep learning, обробка великих даних (Big Data), у статистиці та інше. Є однією з найбільш використовуваних мов програмування завдяки закладеній в неї простоті дизайну, гнучкості та лаконічності [10].

До основних значущих позитивних характеристик мови відносяться:

- лаконічність: більшість проблем, які вирішуються програмістами потребують використання мінімальної кількісті синтаксичних конструкцій Python у порівнянні із проаналізованими в цьому розділі мовами програмування. Вважається [10], що код програми є читабельним та сприймається швидше аніж код, написаний іншими мовами;
- можливість використання статичної типізації засобами додаткових контролерів, таких як MyPy [11]. Це дозволяє вирішити проблеми невідповідності типів перед виконанням програми;
- інтегрованість і розширюваність: ПЗ написане іншими мовами, програмування, наприклад C, C++, Java, може бути застосоване, як частина ПЗ, написаного мовою Python, і навпаки;
- кросплатформеність: завдяки тому, що Python – інтерпретована мова, код, розроблений на цій мові може бути виконаним на будь-якій апаратній платформі, де встановлений інтерпретатор Python;
- багата стандартна бібліотека: вона містить важливі та корисних модулі, які можуть бути застосовані в різних видах проєктів;
- ООП: мова підтримує концепції і підходи, притамані іншим об’єктно орієнтованим мовам програмування. Дозволяє створювати нові типи даних за допомогою класів використовувати множинне наслідування. Однак є нюанси, наприклад відсутність поліморфізму методів класів;
- асинхронність досягається многопоточностю. В Python версії 3.7 або новіше є вбудований механізм асинхронності за допомогою корутин [13].

До недоліків відносяться:

- відсутність інтерфейсів. За рахунок того, що мова створювалася як динамічно типізована, застосування в ній інтерфійсів не можливе;
- низька швидкість роботи високонавантажених додатків за рахунок проблем із внутрішньою оптимізацією інтерпретатора та проблем із очисткою пам’яті [14];
- менша кількість бібліотек у порівнянні із JavaScript.

2.1.3. *Java*

Java є строго типізованою компільованою об’єктно орієнтованою мовою програмування загального призначення, розробленою компанією Sun Microsystems та підтримуваною Oracle [15]. Концепція мови програмування Java – мати якомога менше залежностей реалізації від апаратної платформи. Для досягнення цього додатки запускаються у середовищі віртуальної машини Java (JVM) Це значить, що ПЗ, яке написане та скомпільоване на Java, може працювати без перекомпіляції на всіх платформах, в яких підтримується JVM.

До переваг Java відносяться:

- реалізація чистого ООП, який де-факто є стандартом, на який посилаються автори підручників та посібників під час розгляду шаблонів програмування або ООП в цілому [16];
- кросплатформеність застосунків, написаних цією мовою. Оскільки код буде виконуватися у віртуальній машині, після його компіляції у так званий байт код, він може бути перенесений між ОС та апаратними платформами, і все ще буде виконуватися правильно;
- простий синтаксис, зрозумілий початківцям. Відсутність заплутаного синтаксичного цукру, як наприклад в C++;
- наявність функцій, що запобігають критичним помилкам;

- підтримка старих версій мови, відсутність критичних змін у нових версіях мови, які викликають проблеми із зворотньою сумісністю;

До недоліків Java відносяться:

- через те, що Java проєктувалася в часи одноядерних процесорів, вона не може використовувати переваги сучасних багатоядерних систем;
- Сильна прив’язка до ООП: реалізація бізнес-логіки та операцій із даними в Java потребує створення класів;
- відсутність функціонального програмування та замикань. Операції із даними мають бути описані структурно;
- відсутність єдиної багатої бібліотеки сторонніх модулів та зручного механізму контролю залежностей, як NPM у JavaScript;
- використання Java 8 і більш нових версій у комерційних цілях є платним;
- низька продуктивність через використання JVM та некоректну роботу прибиральника сміття;
- асинхронність досягається за рахунок використання важких з точки зору ОС процесів та потоків.

2.1.4. *JavaScript*

JavaScript – динамічно типізована прототипова скриптова мова програмування, яка відповідає стандарту ECMAScript [17] і призначена у першу чергу для реалізації інтерактивної взаємодії користувача із веб-сторінкою. Мову класифікують як прототипну, тобто множину об’єктно-орієнтованої мови. Наслідування відбувається без існування поняття класу, а за допомогою існуючого примірника об’єкту – прототипу [18]. Для забезпечення роботи ПЗ написаного цією мовою на стороні серверу у 2009 році було створено фреймворк Node.js.

Node.js – це програмна платформа на рушії V8, який розробляє

компанія Google для свого інтернет-браузеру Google Chrome [19]. Вона додає до JavaScrtipt можливості для взаємодії із низькорівневим АПІ ОС, таким як ввід-вивід, робота із мережею та інше. Таким чином, при використанні Node.js та JavaSctipt з'являєтсья можливість застосовувати останній в якості серверної мови програмування для високонавантажених додатків. Основне місце застосування Node.js – розробка веб-серверів, активно взаємодіючих із БД, файловою системою, мережею або веб-сокетами, без виконання важких обчислень, таких як статистичні чи математичні.

Основні переваги JavaScript:

- мова JavaScript є мультипарадигменою, однак найбільш поширеною концепцією є функціональна парадигма програмування. Цей стиль передбачає побудову програми, при якому процес виконання уявляється як результат обчислення математичних функцій, які при цьому уникають наскільки це можливо зміни стану даних (мутації);
- асинхроність за рахунок черги подій. Код мовою JavaScript виконується рушієм V8, розробленим компанією Google. Існує поняття черга подій, де всі виклики функцій потрапляють до черги і чекають, доки рушій викличе їх для отримання результату. Існують підсписники на події виконання функцій, які очікують на результат та будуть сповіщені рушієм, коли цей результат буде доступним. Таким чином, програма, написана мовою JavaScript є однопоточною: проте асинхроною, оскільки ресурсомісткі операції, такі як ввід-вивід та читання із диску, не блокують потік виконання [20];
- гарна оптимізація рушія V8, що робить програми на JavaScript дуже швидкими у порівнянні із Python [21];
- наявність великої бібліотека модулів. Спільнота JavaScript є однією із найбільших спільнот у світі за даними компанії

JetBrains [22]. Логічно, що така спільнота створює велику кількість модулів для усіх можливих потреб і сценаріїв використання мови JavaScript;

- зручні менеджери пакетів. Для контролю версій використовуваних у проєкті залежностей створено менеджер пакетів NPM [23]. За суб'єктивною оцінкою, вони є одними із найзручніших пакетів серед усіх, які використовуються в представлених у цьому порівнянні мов програмування;
- використання динамічна типізації, за рахунок чого розробка ПЗ є дуже швидкою та не потребує глибоких знань цієї мови. Поріг входу початківців є низьким, у порівнянні із іншими мовами.

Очевидними недоліками JavaScript та Node.js є:

- не строга типізація: в момент виконання програми може з’явитися проблема невідповідності типів та інтерфейсів, що призведе до неочікуваної поведінки програми або її аварійного завершення;
- нижча за Go продуктивність: оскільки код виконується інтерпритатором а не на пряму процесором, його швидкодія нижча, аніж в інших роглянутих мовах;
- однопоточність: ПЗ запускається в одному потоці, де за асинхроність відповідає рушій та черга подій. Завдяки швидкому перемиканню між подіями складається враження паралелізму. Однак якщо окрема подія потребує великого проміжку часу на обчислення, весь потік виконання зупинеться. Окрім цього в системах на основі багатоядерних ЦП більшість ядер буде простоювати, оскільки JavaScript не може розподілити навантаження між ними. Для рівномірного навантаження на систему має бути застосована кластерізація програми – запуск на виконання декількох незалежних екземплярів програми;

- відсутність ООП у загальному розумінні: оскільки JavaScript – протипна мова, в ній відсутній поліморфізм, множинне наслідування, перевантаження методів та інші поняття, притаманні об’єктно-орієнтованим мовам програмування.

2.1.5. *TypeScript*

TypeScript – компільована, мультипарадигмена мова програмування, яку розробляє компанія Microsoft з 2012 року [24]. Ідея створення цієї мови полягає в усуненні проблем, які притаманні JavaScript, а саме статична строга типізація, відсутність реалізації принципів ООП, таких як наслідування, інтерфейси та узагальнення. Причиною до проєктування цієї мови стали проблеми, з якими стикається команда інженерів програмного забезпечення при роботі із масштабними проєктами корпоративного рівня.

Код на мові TypeScript компілюється в код на мові JavaScript, що дозволяє застосовувати його в оточенні, де може виконуватися мова JavaScript. Крім цього TypeScript дозволяє використовувати модулі, написані мовою JavaScript. Таким чином велика бібліотека модулів NPM доступна для використання в програмі на TypeScript.

Всі переваги та недоліки JavaScript притаманні TypeScript, окрім проблем із типізацією та нюансами реалізації ООП.

2.1.6. *Висновки*

Проаналізувавши популярні мови програмування розроблення серверної частини веб-додатку (табл. 2.1), прийшли до висновків, що найбільш доцільним для даного веб-застосунку є застосування саме Node.js на мові програмування TypeScript, яка не тільки відповідає всім встановленим вимогам, а й дозволяє використати деяку частину реалізованих алгоритмів на цій мові у клієнтській частині системи.

Таблиця 2.1

Порівняння мов програмування для розроблення серверної частини

Мова \ Критерії	Наявність інструментів функціональної парадигми	Наявність жорсткої типізації	Асинхронна модель виконання коду	Наявність фреймворку для побудови веб-серверу	Використання парадигми ООП
Go	+	+	+	+	–
Python	+	+/–	+	+	+/–
Java	–	+	+/–	+	+
JavaScript	+	–	+	+	+/–
TypeScript	+	+	+	+	+

2.2. Вибір технології для розроблення клієнтської частини

В останні роки основною мовою розробки інтерактивних веб-сайтів став JavaScript, який де-факто став домінуючим інструментом для веб-розробки [25]. Тому на питання, яку мову програмування обрати для створення клієнтського інерфейсу можна дати відповідь – JavaScript.

На початку еру веб-сайтів, вони були простими та могли у більшості свої відображати лише ту інформацію, яку отримали від веб-серверу. З плином часу додатки ставли більш складнішими та потребували більше обчислень на стороні клієнта. З появою технології AJAX [26], яка дозволила робити асинхроні запити до веб-серверу після завантаження сторінки у вікно браузеру, додатки стали дедалі більше схожими на повноцінні настільні застосунки. Для вирішення проблемі із зрослої складності структури проєктів та маніпуляції із даними було створено фреймворки до

мови JavaScript. Зараз серед веб-розробників постає питання вибору серед трьох їх основних представників: React.js, Angular, Vue.js [27]. Основні критерії вибору наступні:

- кількість бібліотек та модулів, доступних до використання;
- простота та лаконічність коду;
- швидкість рендеру веб-сторінки та оптимізація роботи із пам’яттю.

2.2.1. *Vue.js*

Vue.js – прогресивний веб-фреймворк для побудови клієнтських інтерфейсів та SPA [28]. Основними перевагами є:

- найменший розмір ядра бібліотеки, у порівнянні із іншими розглянутими фреймворками [29];
- представлення інтерфейсів у виді компонентів, окремо виділяючи в кожному його блоки шаблону мовою HTML, скриптів мовою JavaScript та стилів мовою CSS. Це зручно тим, що дозволяє уникати проблем із змішуванням різних мов та інструментів, що притаманне вихідному коду проєктів на React;
- реалізація віртуального DOM – більш легковісного аналогу DOM-дерева браузеру. Фреймворк на момент рендеру перевіряє, які вузли змінилися в віртуальному DOM і синхронізує їх із браузерним DOM. Це дозволяє мінімізувати кількість операцій перемальовування вікна браузера;
- за результатами тестування, є найшвидшим серед усіх фреймворків [30], що є дуже добрим для UX.

В той самий час недоліками є:

- відносно новий гравець на ринку веб фреймворків. Перша версія Vue.js була представлена у 2016 році. Має меншу кількість проєктів, які його використовують аніж інші фреймворки;

- підтримується ентузіастами, не спонсується жодною великою компанією, тому швидкість виходу нових мажорних версій є нижчою, за інші фреймворки;
- для побудови додатку за моделлю MVC слід використовувати сторонні бібліотеки для контролювання стану застосунку.

2.2.2. *Angular*

Angular – клієнтський фреймворк для побудови односторінкових веб-додатків [31]. В якості основного архітектурного шаблону Angular використовує підхід Model-View-Controller (MVC). Розробляється компанією Google, перша версія представлена у 2010 році.

Перевагами Angular є:

- більшість необхідних для комфортної роботи модулів вже є вбудованими в стандартну бібліотеку;
- синтаксис фреймворка заохочує до реализації архітектури за принципом MVC. Що робить його незамінним при розробці великих систем корпоративного рівня.
- розробка ведеться виключно на мові програмування TypeScript, що виключає проблеми, викликані динамічною не строгою типізацією JavaSctipt.

Недоліками є:

- великий об’єм ядра бібліотеки, що сповільнює завантаження додатків на мобільних приладах;
- найнижча серед інших фреймворків швидкість роботи та первинної загрузки і ініціалізації, що погано позначається на UX та SEO оптимізації [29];
- жорстка прив’язаність на модель MVC, не підходить для реалізації MVP продукту, через потребу у більшій кількості часу на розробку та проєктування.

2.2.3. *React.js*

React.js – бібліотека, написана на мові JavaScript, призначена для побудови користувацьких інтерфейсів [34]. Пропонує компонентний підхід до побудови користувацького інтерфейсу, де кожен функціональний чи візуальний елемент предствляє собою окремий модуль із власним станом і поведінкою, який поєднує в собі і розмітку і логіку роботи . Для створення структури веб-сторінки використовується JSX – розширення HTML, який дозволяє використовувати код мовою JavaScript безпосередньо в середині XML дерева - розмітки. React розробляється та спонсується компанією Facebook, оскільки він був створений для підтримки веб-сторінок однойменної соціальної мережі.

До переваг React.js відносяться:

- добре оптимізований Three Shaking. Це технологія, за допомогою якої браузер розуміє, яка частина коду додатку зараз потрібна та завантажує лише її, тим самим він економить Інтернет-трафік пристрою;
- співтовариство користувачів цієї бібліотеки вимірюється тисячами розробників та є одним із найбільших у світі [33];
- найменший розмір ядра бібліотеки, розмір файлу react.js, який має бути завантажений кожним додатком;
- використання віртуального DOM-дерева: як і Vue.js, React.js оптимізує операції зміни реального DOM за рахунок знаходження змін у стані віртуального дерева. Час маніпуляцій із віртуальним DOM є мізерно малим відносно маніпуляцій зі справжнім DOM, отже такий підхід дозволяє отримати величезний приріст у швидкості оновлень стану веб-додатку;
- використання JSX: компоненти React.js є дуже малими у порівнянні із компонентами Vue.js та Angular. Вони поєднують в собі лаконічність та добру читабельність, що пришвидшує процес розробки ПЗ;

- найвища популярність серед інших фреймворків: за даними StackOverflow [21] має найбільшу;
- використання функціональних компонентів та React Hook API [34] дозволяє розроблювати клієнтський інтерфейс за парадигмою функціонального програмування.

2.2.4. *Висновки*

Зважаючи на результати проведеного аналізу та описані переваги, які надає фреймворк React.js у порівнянні із аналогічними рішеннями, для розроблення клієнтського інтерфейсу веб-додатку було обрано його.

2.3. Вибір СУБД

Важливою частиною ПЗ, що забезпечує надійне зберігання даних, якими оперує програма, та надає доступ до них, є база даних. СУБД – сукупність засобів загального або спеціального призначення, що забезпечують управління створенням і використанням баз даних [35].

2.3.1. *Вибір моделі СУБД*

Одним із видів класифікації СУБД є класифікація за моделлю зберігання та представлення даних. За цією характеристикою виділяють реляційні (SQL) та документо-орієнтовані бази даних (NoSQL). SQL – декларативна мова запитів, що застосовується для створення, модифікації та управління даними в реляційній БД [37]. Різниця між ними полягає в обмеженнях та перевагах, які вони надають для своїх користувачів.

В нереляційних БД зберігають неструктуровані дані, які зазвичай представляють собою документи, наприклад у форматі JSON, або у вигляді ключ-значення. В реляційних БД зберігають структуровані дані із зумовленими зв'язками між ними.

Перевагами NoSQL БД є:

- висока швидкість роботи із необробленими «сирими» даними;

- підтримка розподіленої моделі зберігання даних та горизонтального масштабування, коли підвищення продуктивності системи досягається не збільшенням обчислювальних ресурсів, доступних кожному екземпляру БД, а збільшенням кількості самих екземплярів БД, наприклад залучаючи додаткові обчислювальні сервери;
- відсутність жорсткої прив’язки до схеми даних;
- завдяки цьому NoSQL БД набули популярності при обробці великих об’ємів даних, або при роботі із розрідженими масивами інформації, наприклад статистикою інтернет реклами [38].

Перевагами реляційних БД є:

- можливість зберігати структуровані дані, які відповідають встановленій схемі;
- підтримка запитів мовою SQL;
- реалізації інкапсуляції окремих операцій із даними в рамках транзакцій;
- можливість виконання міграцій даних та їх схеми;
- підтримка відношень між даними та можливість отримувати зв’язані дані в рамках одного запиту.

Зважаючи на це, SQL БД обирають для ПЗ, яке має справу із структурованими консистентними даними, із виділеними відношеннями між ними або при роботі із високонавантаженими системами, які мають проблему із конкуруючими запитами, що вирішується використанням транзакцій для їх ізоляцій та пріоритезації. Прикладами таких систем є банки та системи CRM.

Для розробки веб-додатку для сприяння пошуку втрачених речей було зроблено вибір на користь реляційних БД для постійного зберігання даних програми та користувачів, оскільки схема використовуваних даних заздалегідь відома та передбачає використання транзацій для взаємодії із . Для короткотривалого зберігання даних із підтримкою швидкого доступу до

них, було використано СУБД ключ-значення Redis [39].

2.3.2. *Вибір ядра СУБД*

Після обрання виду СУБД постає питання про обрання ядра БД. Оскільки вони реалізують стандарт мови SQL, різниця між ними полягає у наданні додаткових можливостей: підтримці зберігання неструктурованих даних (JSON), реалізації мови написання власних функцій для роботи із даними, підтримці повнотекстового пошуку.

До ядра БД було висунуто такі вимоги:

- вільне розповсюдження (безкоштовність);
- підтримка сервісом AWS RDS [40];
- можливість зберігати неструктуровані дані у форматі JSON;
- відповідність вимогам ACID;
- підтримка індексів.

2.3.3. *MySQL*

MySQL – система управління реляційними базами даних, яку розроблює та вільно розповсюджує компанія Oracle [41]. Написана мовами C та C++, СУБД MySQL є сумісною з усіма основними операційними системами.

Особливостями СУБД MySQL є:

- відмінність реалізація синтаксису MySQL від стандарту SQL є мінімальною;
- підтримка зберігання неструктурованих даних у форматі JSON.
- наявність транзакцій, їх рівнів ізоляції та функції відміни;
- наявність тригерів, збережуваних процедур та представлень;
- підтримується основними бібліотеками ORM для мови програмування Typescript в оточенні Node.js.

До відомих вад MySQL відносять [42]:

- висока складність горизонтального масштабування;

- низька швидкодія при обчисленні комплексних запитів.

2.3.4. *PostgreSQL*

PostgreSQL – відкрита реляційна СУБД, написана на мові програмування C [43]. Була одною з перших СУБД, що з'явилися на ринку, має широку розповсюдженість та широку спільноту користувачів.

Визначними перевагами СУБД PostgreSQL є:

- еталона реалізація повнотекстовий пошуку серед реляційних баз даних [42];
- зручна для використання реалізація горизонтального масштабування;
- реалізація матеріалізованих представлень даних;
- підтримка операції каскадного видалення таблиці даних;
- ширший спектр доступних видів індексів даних: hash, b-tree, GiST, SP-GiST, BRIN і Bloom;
- зберігання неструктурованих даних у текстовому (JSON) та бінарному (JSONB) форматі.

Відомими недоліками PostgreSQL є:

- часткова підтримка стандарту ANSI SQL;
- має проблеми із падінням швидкодії при виконанні запитів із багатьма операторами групування.

2.3.5. *Висновки*

В результаті проведеного аналізу було встановлено, що розглянуті СУБД відповідають встановленим вимогам до ядра БД. Однак для реалізації веб-додатку будо обрано СУБД PostgreSQL, через її більшу зручність при маніпулюванні неструктурованими даними, повній підтримці повнотекстового пошуку та матеріалізованих представлень даних.

3. СТРУКТУРНО-АЛГОРИТМІЧНА ОРГАНІЗАЦІЯ

3.1. Загальний опис системи

В розробці програмного забезпечення важлива роль відводиться процесу первинного аналізу та плануванню самого процесу розробки [30]. На цьому етапі перед командою інженерів стоїть завдання зібрати та описати вимоги, які висуваються до готового програмного продукту. Для більш глибокого розуміння специфіки предметної області, в якій працюватиме додаток слід визначити його цілі та проблематику, для вирішення якої він розробляється.

Веб-додаток для сприяння пошуку втрачених речей представляє собою веб-сайт, який дозволяє своїм відвідувачам спілкуватися за допомогою чату із людьми, які могли знайти їх речі. Авторизовані користувачі мають змогу отримувати посилання на сторінку із чатом з ними. Посилання закодовано у вигляді зображення QR-коду. Це зображення рекомендовано розміщувати на власних речах користувачів будь-яким зручним для них способом, наприклад у вигляді наліпки або брелку. Якщо річ із такою міткою буде втрачена її може знайти інша особа (ОЯЗЧВ). Вона може перейти за посиланням в зображені та побачити веб-сторінку із інформацією про власника QR-коду. Слід зазначити, що користувач може обрати, яку саме інформацію про себе надати людям, які потрапили на сторінку із нею завдяки посиланню у мітці на втраченій речі. Це може бути його контакти, фото, адреса або текстове повідомлення із проханням повернути знайдену чужу власність. Система запропонує відвідувачу почати спілкування із людиною за допомогою анонімного чату. Щоб це зробити, ОЯЗЧВ повинна надати декілька фото із знайденою річчю. В обмін на них, їй буде згенеровано посилання, за яким буде доступним ініційований ним чат. Це посилання слід зберегти для подальшого користування. Після завантаження фото із речами ВЗР отримає повідомлення від веб-додатку і може відмовитися від спілкування, або погодитися. В останньому випадку він отримає посилання на чат між ним і ОЯЗЧВ. З його допомогою вони

можуть дійти згоди у діалозі про повернення втраченої власності її власнику, або ні. Вся відповідальність за це лежить виключно на них, адміністрація веб-додатку не втручається в хід перемовин.

3.1.1. *Аналіз проблематики та цілей веб-додатку*

На основі загального опису веб-додатку було сформовано перелік проблеми, для рішення яких він створюється. Рішення цих проблем і створює цінність, яку веб-додаток представляє для своїх користувачів.

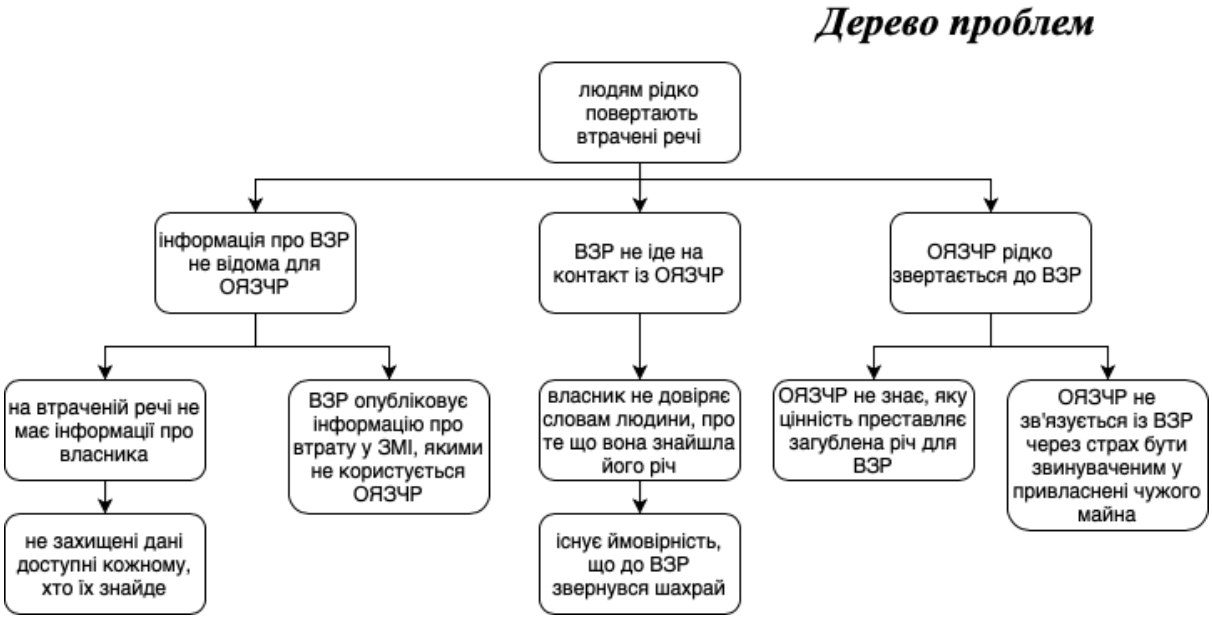


Рис. 3.1. Дерево проблем продукту

До згаданих проблем відносяться:

1. Проблема відсутності персоналізації на загублених речах, що виключає можливість ідентифікувати їх власника та повідомити його про знахідку.
2. Якщо ВЗР опублікує замітку в ЗМІ про свою втрату, він має високу ймовірність стати жертвою шахраїв, котрі під виглядом ОЯЗЧВ спробують отримати гроші за повернення речі, якою не володіють.

3. Відсутність мотивації ОЯЗЧВ телефонувати власнику загубленої речі, оскільки вважає що річ має не високу цінність для власника і може бути безболісно для нього загублена.

Зважаючи на описані проблеми, було побудовано дерево проблем, представлене на рис. 3.1.

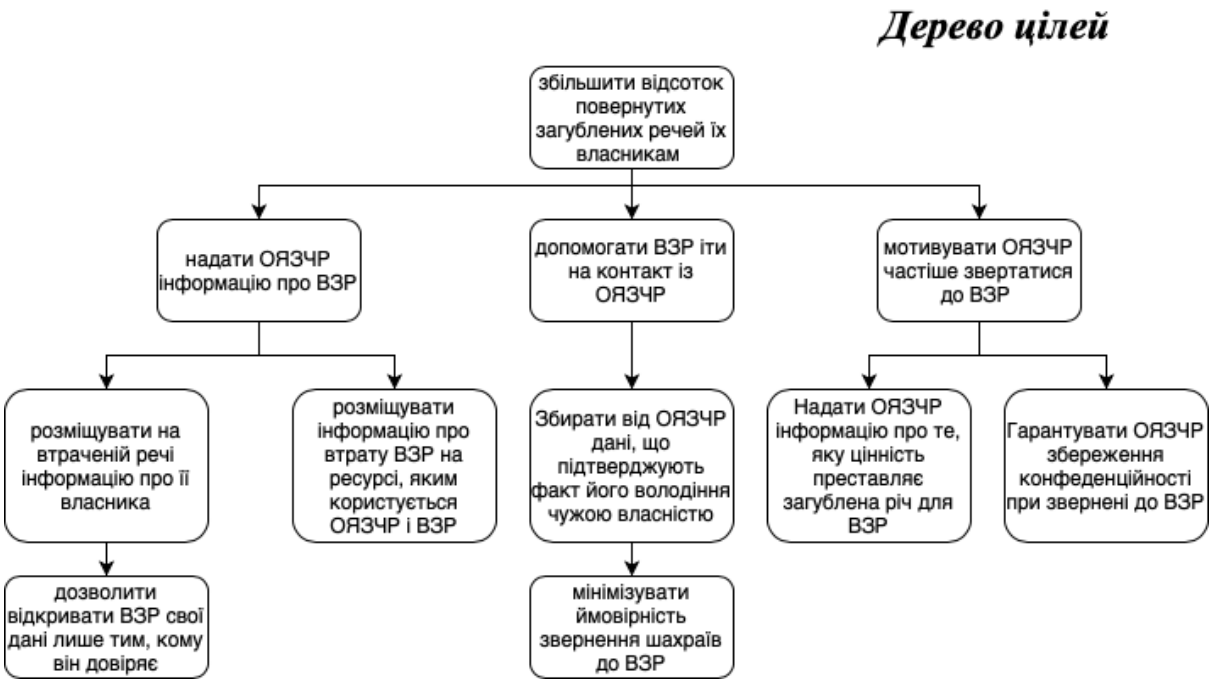


Рис. 3.2. Дерево цілей продукту

Після дослідження проблематики предметної області сформовано цілі, досягнення яких в рамках розроблюваного веб-додатку для сприяння поверненню втрачених речей призведе до їх вирішення. Загальної ціллю є підвищення відсотку повернутих речей їх власникам. Для цього мають бути запропоновані шляхи до вирішення проблеми ідентифікації ВЗР, збереження його конфіденційності та збільшення вмотивованості ОЯЗЧР почати спілкування із власником знахідки. Структура цілей продукту зображена у вигляді дерева цілей (рис. 3.2).

На основі дерева цілей було побудовано дерево результатів, які будуть отримані в процесі створення ПЗ веб-додатку (рис. 3.3).

Провівши дослідження проблемної області веб-додатку для сприяння

поверненню втрачених речей, та сформувавши цілі і результати, які очікуються від реалізації цього додатку, було сформовано перелік вимог, яким має відповідати зазначене ПЗ.

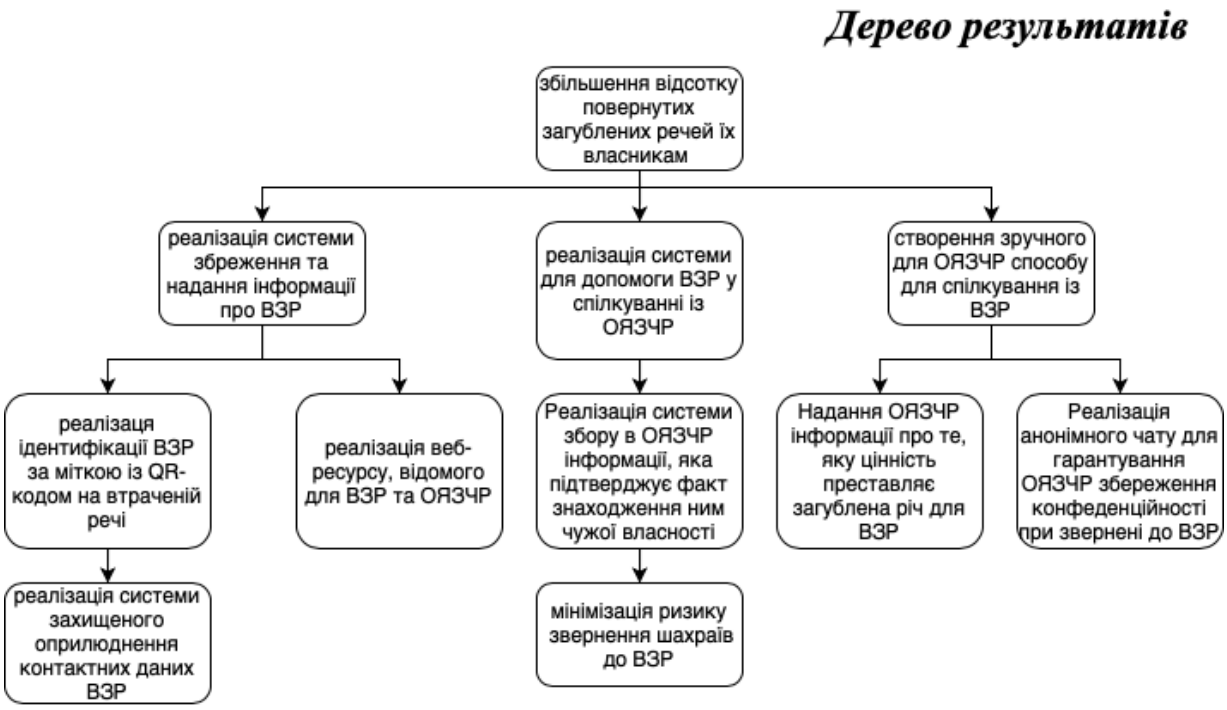


Рис. 3.3. Дерево результатів продукту

3.1.2. *Формулювання вимог до програмного забезпечення*

Одним із визначення поняття «вимога» є специфікація того, що має бути отримано, що описує поведінку системи або її атрибути та властивості. Вимоги можуть бути обмеженнями на процес розробки системи [31]. Виділяють такі види вимог:

- функціональні вимоги:
  - бізнес вимоги: містять високорівневий опис цілей замовників розроблення програмного рішення;
  - вимоги користувача: описують цілі і завдання користувачі, які система вирішує;
- нефункціональні вимоги:
  - зовнішні інтерфейси: опис аспектів взаємодії із іншими системами;

- атрибути якості: додатковий опис функцій продукту, виражений через опис його характеристик;
- обмеження: умови, які обмежують вибір можливих рішень щодо реалізації компонентів системи;
- вимоги до ефективності: встановлення границь використання системою обчислювальних ресурсів при заданому рівні навантаження.

Функціональні вимоги описують необхідні функціональні особливості роботи системи та компонентів, за якими можна робити висновок, про те, чи вирішує система проблеми і досягає цілі, які були поставлені перед нею замовником.

Нефункціональні вимоги напроти описують додаткові характеристики та особливості, якими має володіти система в цілому для того, аби відповідати очікуваному рівню якості та надійності.

Таким чином, на основі опису системи та положення про обов'язкові властивості вимог до ПЗ, було сформовано перелік вимог. Реалізація цих вимог в веб-додатку для сприяння поверненню втрачених речей дозволить вважати його закінченим та таким, що вирішує проблеми, описані в дереві проблем.

Таблиця 3.1

Вимоги до продуктивності

Код вимоги	Опис вимоги
PER -1	Веб-сторінка має ініціалізуватися менше ніж за 3 секунди після завантаження файлі в браузер користувача.
PER-2	Система має відповідати на запити менш ніж за 1 секунду
PER-3	Система має коректно працювати при навантажені в 1000 запитів на секунду.

Опишемо нефункціональні вимоги розроблюваного веб-додатку, згрупувавши її за категоріями: вимоги до продуктивності описані в табл. 3.1, вимоги до безпеки описані в табл. 3.3, вимоги до реалізації в табл. 3.2.

Таблиця 3.2

Вимоги до безпеки

Код вимоги	Опис вимоги
SEC-1	Система має бути захищена від поширених атак типу XSS, CSRF, SQL Injection
SEC-2	Система має зберігати користувачів мають зберігатися у захешованому вигляді, який не піддається зворотній трансформації
SEC-3	Система має бути захищена від атаки повним перебором можливих адресів чатів із ВЗР за допомогою використання сервісу reCAPTCHA від Google

Таблиця 3.3

Вимоги до реалізації

Код вимоги	Опис вимоги
IMP-1	Серверна частина веб-додатку має бути побудована за мікросервісною архітектурою та розгорнута за допомогою AWS Lambda.
IMP-2	Збереження зображень користувачів має виконуватися за допомогою сервісу AWS S3.
IMP-3	Система має підтримувати автоматичну доставку коду та розгортання.
IMP-4	Веб-додаток має коректно працювати в оточенні сучасних браузерів, якими користується 95% людства.

Для формування функціональних вимог, слід спершу описати ключові сценарії користування веб-додатком користувачами за допомогою діаграм прецедентів (англ. “use case diagram”).

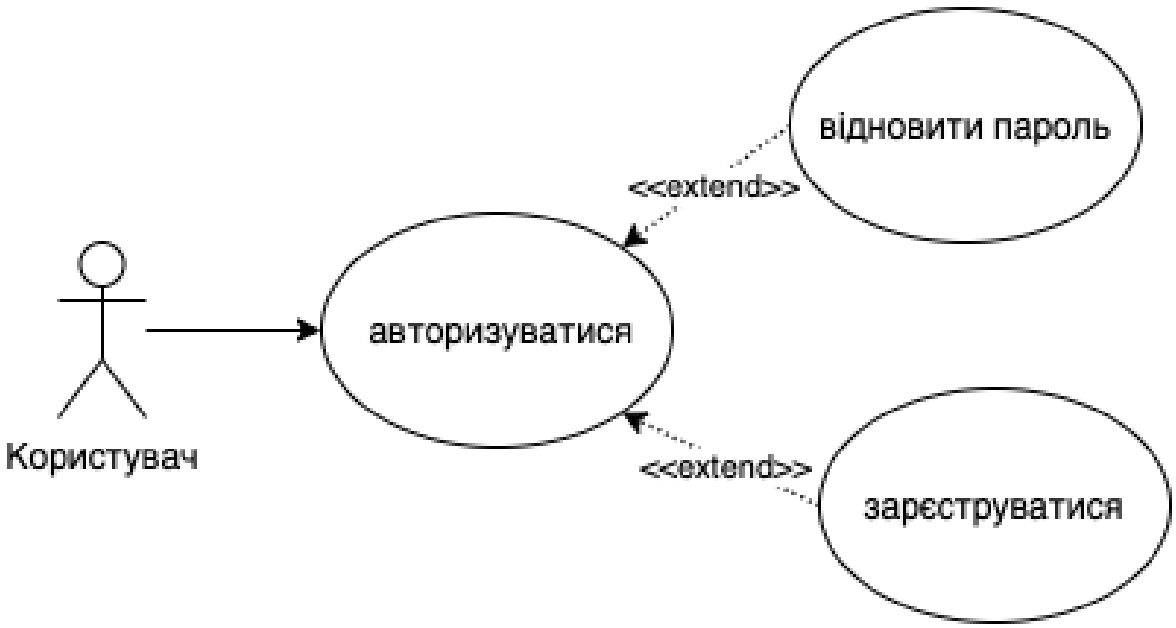


Рис. 3.4. Прецедент: Створення нового облікового запису або авторизація користувача в системі

Прецедент 1: «Створення нового облікового запису або авторизація користувача в системі» (рис. 3.4).

1. Користувач переходить на сторінку із додатком.
2. Користувач відкриває форму авторизації або реєстрації.
3. Користувач додає свої дані.
4. Користувач підтверджує вірність електронної адреси за допомогою листа із посиланням на сторінку підтвердження, якщо він тільки реєструється.
5. Якщо користувач авторизується та забув свій пароль, він може надіслати запит на відновлення паролю та отримати листа на вказану в обліковому записі електрону адресу листа із посиланням на сторінку зміни паролю.

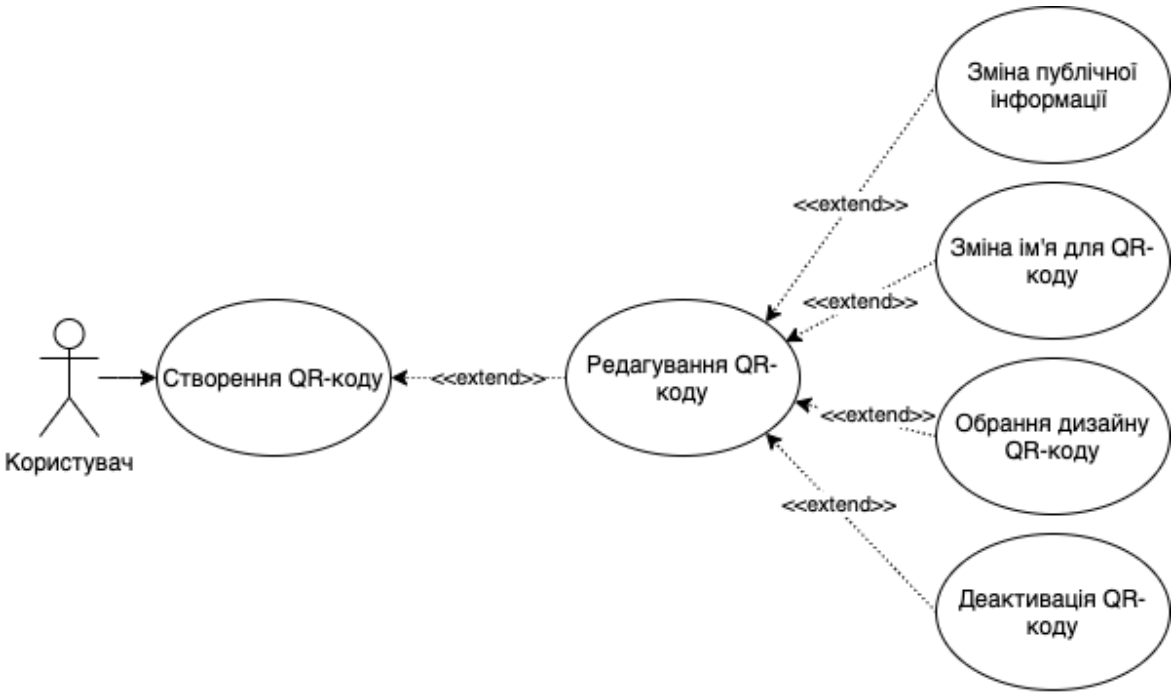


Рис. 3.5 Прецедент: Створення нового облікового запису або авторизація користувача в системі

Прецедент 2: «Створення або редагування QR-коду» (рис. 3.5).

Передумова: користувач авторизований.

1. Користувач знаходиться на сторінці створення QR-коду.
2. Користувач вибирає інформацію, яка буде доступна для відвідувача, який перейшов за посиланням в QR-коді.
3. Користувач обирає дизайн QR-коду.
4. Користувач зберігає отриманий QR-код в системі.
5. Користувач за потреби деактивує QR-код.
6. Користувач зберігає зображення QR-коду на локальний комп'ютер та роздруковує його.

Прецедент 3: «Перегляд запиту на початок спілкування» (рис. 3.6).

Передумова: користувач авторизований, користувач має існуючий QR-код.

1. Користувач знаходиться на сторінці перегляду бесід із ОЯЗЧВ.
2. Користувач обирає запит на початок бесіди.

3. Користувач бачить фотографії, які до заявки прикріпила ОЯЗЧВ, та геопозицію, звідки надійшов запит.
4. Користувач погоджується та продовжує бесіду із ОЯЗЧВ або блокує її.

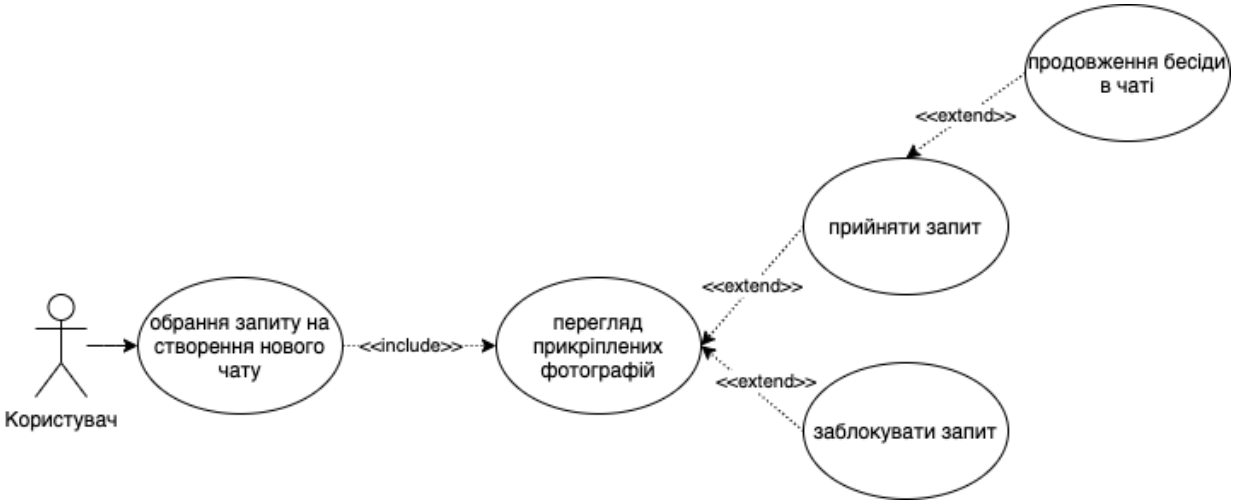


Рис. 3.6. Прецедент: Перегляд запиту на початок спілкування

Прецедент 4: «Ведення бесіди із ОЯЗЧВ» (рис. 3.7).

Передумова: користувач авторизований, користувач прийняв запит на створення бесіди із ОЯЗЧВ.

1. Користувач знаходиться на сторінці бесіди із ОЯЗЧВ.
2. Користувач бачить повідомлення в цій бесіді.
3. Користувач може надіслати повідомлення ОЯЗЧВ.
4. Користувач отримує нотифікації про нові повідомлення від ОЯЗЧВ на електрону адресу.
5. Користувач може заблокувати бесіду, якщо вважає її завершеною.
6. Користувач може надати свою контактну інформацію, збережену у системі.
7. Користувач бачить інформацію про ОЯЗЧВ, яку вона надала.

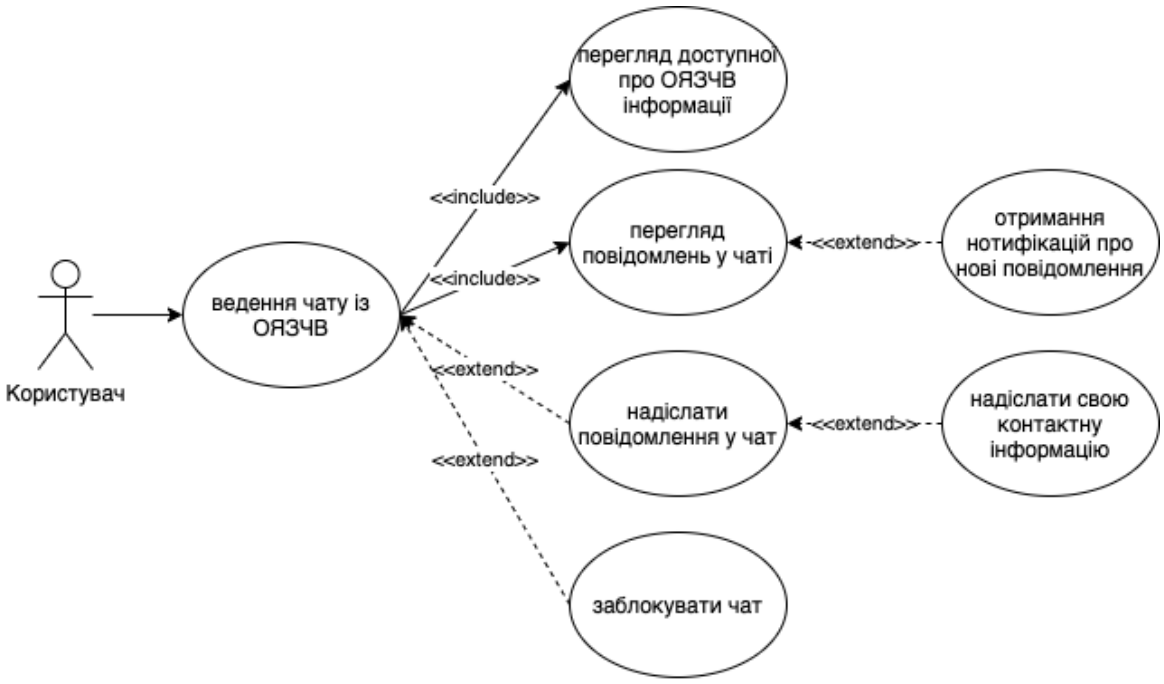


Рис. 3.7. Прецедент: Ведення бесіди із ОЯЗЧВ

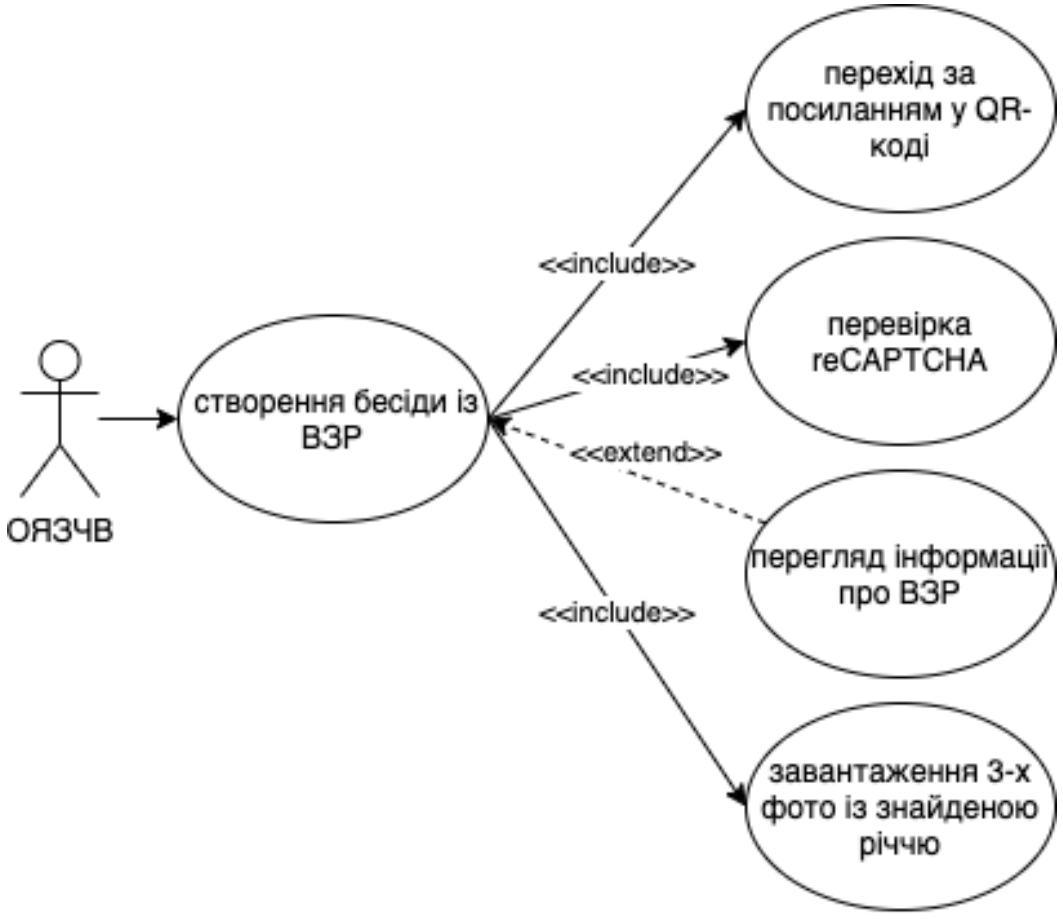


Рис. 3.8. Прецедент: Створення запиту на чатування із ВЗР від ОЯЗЧВ

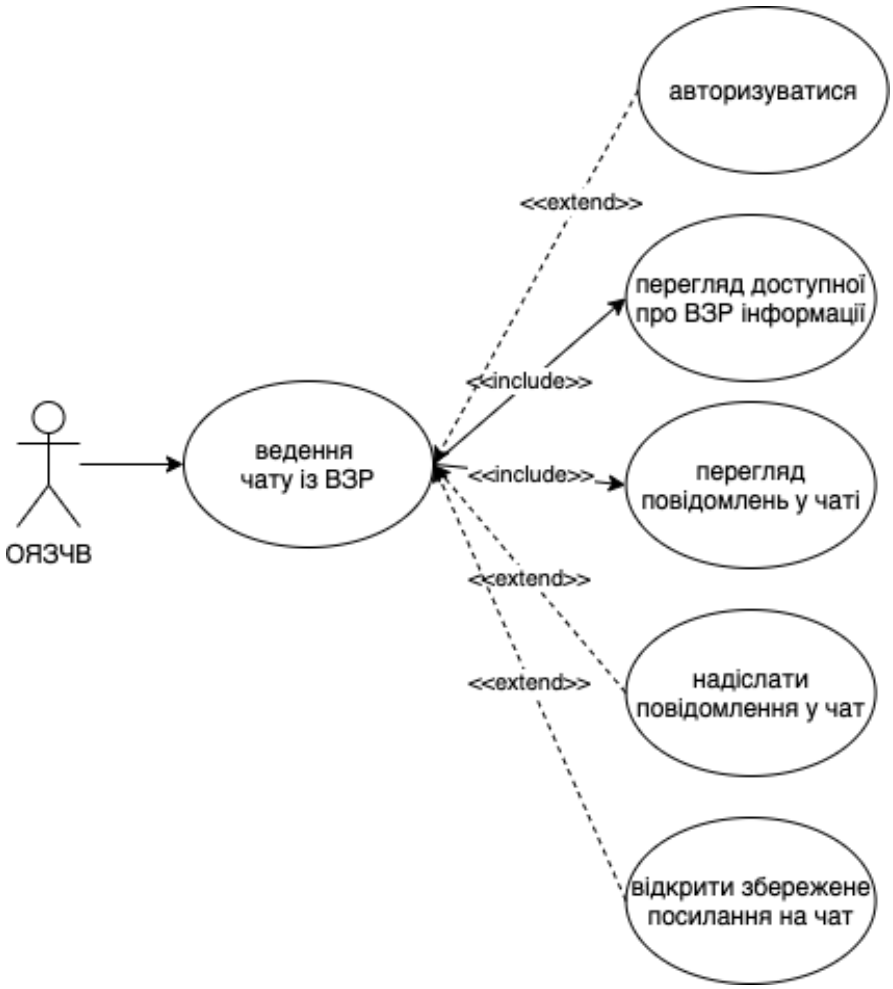


Рис. 3.9. Прецедент: Ведення бесіди із ВЗР

Прецедент 5: «Створення запиту на чатування із ВЗР від ОЯЗЧВ» (рис. 3.8).

Передумова: ОЯЗЧВ знайшов QR-код ВЗР та перейшов за посиланням в ньому.

1. ОЯЗЧВ проходить перевірку reCAPTCHA.
2. ОЯЗЧВ бачить інфомацію, яку про себе надав ВЗР для обраного QR-коду.
3. ОЯЗЧВ може авторизуватися або зареєструватися, або залишитися анонімним.
4. ОЯЗЧВ пише повідомлення до ВЗР, і прикріплює 3 фотографії знайденої речі.

5. ОЯЗЧВ отримує нове посилання, за яким він може завжди потрапити на бесіду із обраним ВЗР, навіть не авторизуючись.

Прецедент 6: «Ведення бесіди між ВЗР та ОЯЗЧВ» (рис. 3.9).

Передумова: ОЯЗЧВ ініціював чат із ВЗР, ВЗР прийняв запрошення.

1. ОЯЗЧВ перейшла на сторінку чату за посиланням.
2. ОЯЗЧВ надсилає повідомлення ВЗР.
3. Якщо ОЯЗЧВ авторизована, вона може надіслати свої дані, які зберігаються в системі.
4. ОЯЗЧВ бачить інформацію про співрозмовника, яку він надав.
5. ОЯЗЧВ може закінчити бесіду.

На основі описаних прецедентів, було сформовано список функціональних вимог (табл. 3.4).

Таблиця 3.4

Функціональні вимоги до веб-додатку

Код вимоги	Зміст вимоги	Атрибути	
		Пріоритет	Складність
F1	Реалізація реєстрації користувачів за допомогою електронної адреси та паролю	Високий	Середня
F2	Можливість відновлювати пароль за допомогою листа на електрону адресу користувача	Середня	Середня
F3	Реалізація авторизації користувачів за допомогою електронної адреси та паролю	Високий	Середня
F4	Перегляд існуючих QR-кодів авторизованим користувачем	Високий	Низька

Продовження табл. 3.4

F5	Створення нового QR-коду авторизованим користувачем	Високий	Середня
F6	Редагування авторизованим користувачем власного існуючого QR-коду	Низький	Середня
F7	Деактивація авторизованим користувачем свого QR-коду	Середній	Середній
F8	Перегляд авторизованим користувачем списку активних чатів	Високий	Середній
F9	Перегляд ВЗР списку заблокованих чатів	Середній	Низький
F10	Відправка ВЗР нотифікації про нове повідомлення в чаті	Середній	Низька
F11	Перегляд ВЗР користувачем повідомлень у чаті	Висока	Висока
F12	Відправлення ВЗР повідомлення у чат із ОЯЗЧВ	Висока	Висока
F13	Блокування ВЗР чату із ОЯЗЧВ	Низький	Середня
F14	Перегляд ВЗР запиту на створення чату із прикріпленим до нього фотографіями від ОЯЗЧВ	Високий	Середня
F15	Кодування в QR код унікального посилання на сторінку перегляду публічної інформації про ВЗР	Високий	Низька
F16	Відображення публічної інформації про користувача ОЯЗЧВ і перейшла за посиланням	Високий	Середня

Продовження табл. 3.4

F17	Перевірка відвідувача сторінки за допомогою Google reCAPTCHA перед відображення йому форми для створення запиту на ініціацію чату із ВЗР	Середній	Низька
F18	Можливість ОЯЗЧВ прикріпити фото знайденої речі до чату	Високий	Висока
F19	Генерація посилання на чат для неавторизованого доступу ОЯЗЧВ	Високий	Низька
F20	Відкриття чату із ВЗР за посиланням на нього для ОЯЗЧВ	Високий	Середня
F21	Можливість ОЯЗЧВ додати існуючий чат до себе в профіль за посиланням на нього після автентифікації в системі	Середній	Середня
F22	Відправка повідомлення від ОЯЗЧВ до ВЗР у чаті	Висока	Низька
F23	Перегляд ОЯЗЧВ повідомлень у чаті із ВЗР	Висока	Низька

3.2. Структурна організація програмного забезпечення

3.2.1. *Обрання типу архітектури ПЗ*

Довгий час основним методом розгортання власного веб-додатку була оренда виділеного серверу, на який слід було встановити ОС та налагодити міжсерверну взаємодію через Інтернет, налогодити процес розгоротання нових версій додатку та інш. Рішенням вищезгаданих проблем може слугувати використання технології безсерверних обчислень, зокрема FaaS («функція як послуга») [44]. Це дозволяє бізнесу та розробникам ПЗ

абстрагуватися від деталей апаратних складових веб-серверів та підтримки їх працездатності. Компанія-постачальник послуг FaaS бере на себе відповідальність за обслуговування дата-центрів і надає зручний інтерфейс для взаємодії з їх ресурсами своїм клієнтам. Це дозволяє економити значні кошти на інфраструктурі проєкту та спеціалістах, які її обслуговують.

Зазвичай FaaS є невід'ємною частиною платформ хмарних обчислень, де відіграє роль основи для побудови додатків. Ідея FaaS з'явилась через часте виникнення проблемних ситуацій, з якими стикаються розробники веб-додатків. Так більшість часу веб-сервер відповідного веб - додатку може не використовувати на 100% ресурси апаратної платформи, на якій він працює. Однак бізнес-власник додатку має платити за періоди часу, коли ресурси не задіяні, так само, як і в період високого навантаження. Крім цього, в моменти раптових піків навантаження продуктивність серверів є обмеженою, а отже і ПЗ не може функціонувати як очікувалося, доки не будуть залучені додаткові обчислювальні потужності. Рішенням наведених проблем слугує використання безсерверних обчислень.

Вираз «безсерверні обчислення» не до кінця передає свою суть, яка може спершу спасти на думку, оскільки сервер, з яким взаємодіє віддалений користувач, насправді існує, однак він не має виділених ресурсів та завжди оперує лише необхідним мінімальним обсягом обчислювальних потужностей, які йому потрібні в кожен момент часу. Зрештою це дозволяє платити лише за ті ресурси, які фактично використовуються, та залучати додаткові обчислювальні потужності в короткі терміни для забезпечення горизонтального масштабування, що є вагомою перевагою такого методу розгортання ПЗ.

До переліку постачальників послуг FaaS відносяться GCP Cloud Functions [45], Azure Function [46] та AWS Lambda [47]. В рамках даного дослідження вибір основи для побудови ПЗ було зроблено на користь AWS через популярність даної платформи, простоту та вичерпність її документації, гарні відгуки з боку розробників-користувачів

цього рішення.

Для того, аби контейнер з ПЗ міг без перешкод масштабуватися в обидва напрямки, пропонується переорієнтація монолітної архітектури системи на сервіс-орієнтовану [48]. Це передбачає виділення функціональних слабкозв'язаних модулів, мікросервісів, кожен з яких несе відповідальність лише за окрему частину функціональності та взаємодіє з іншими через їх зовнішні інтерфейси.

Рекомендаціями до реалізації веб-додатку, побудованого на основі моделі FaaS є:

- розроблення системи на основі мікросервісної архітектури;
- відмова від використання сторонніх сервісів збереження логів, БД, CDN, надання переваги стандартним для обраної платформи рішенням для цих потреб;
- використання принципу CD при проєктуванні процесу розгортання додатку;
- використання фасаду над мікросервісною архітектурою для інкапсуляції від зовнішнього світу особливостей реалізації.

3.2.2. *Функціональні модулі веб-додатку*

На основі міркувань із підпункту 3.2.1 розроблюваний додаток для сприяння повернення втрачених речей було вирішено розроблювати із використанням сервіс-орієнтованого підходу. Для забезпечення відповідності розроблюваного ПЗ висунутим до нього вимогам, було виділено наступні функціональні модулі (рис. 3.10) кожен з яких є окремою програмою, яка може бути виконана в оточенні сервісу AWS Lambda. Проаналізуємо представлені сервіси більш детально.

Хостинг статичної веб-сторінки SPA відбувається за допомогою сервісу AWS CloudFront [49] – сервіс глобальної доставки контенту (англ. "CDN"). З його допомогою користувачі отримують останні версії сторінок веб-сайту із мінімальною затримкою. Також він дозволяє

приєднати до веб-додатку домене ім’я за яким він є доступним користувачам.

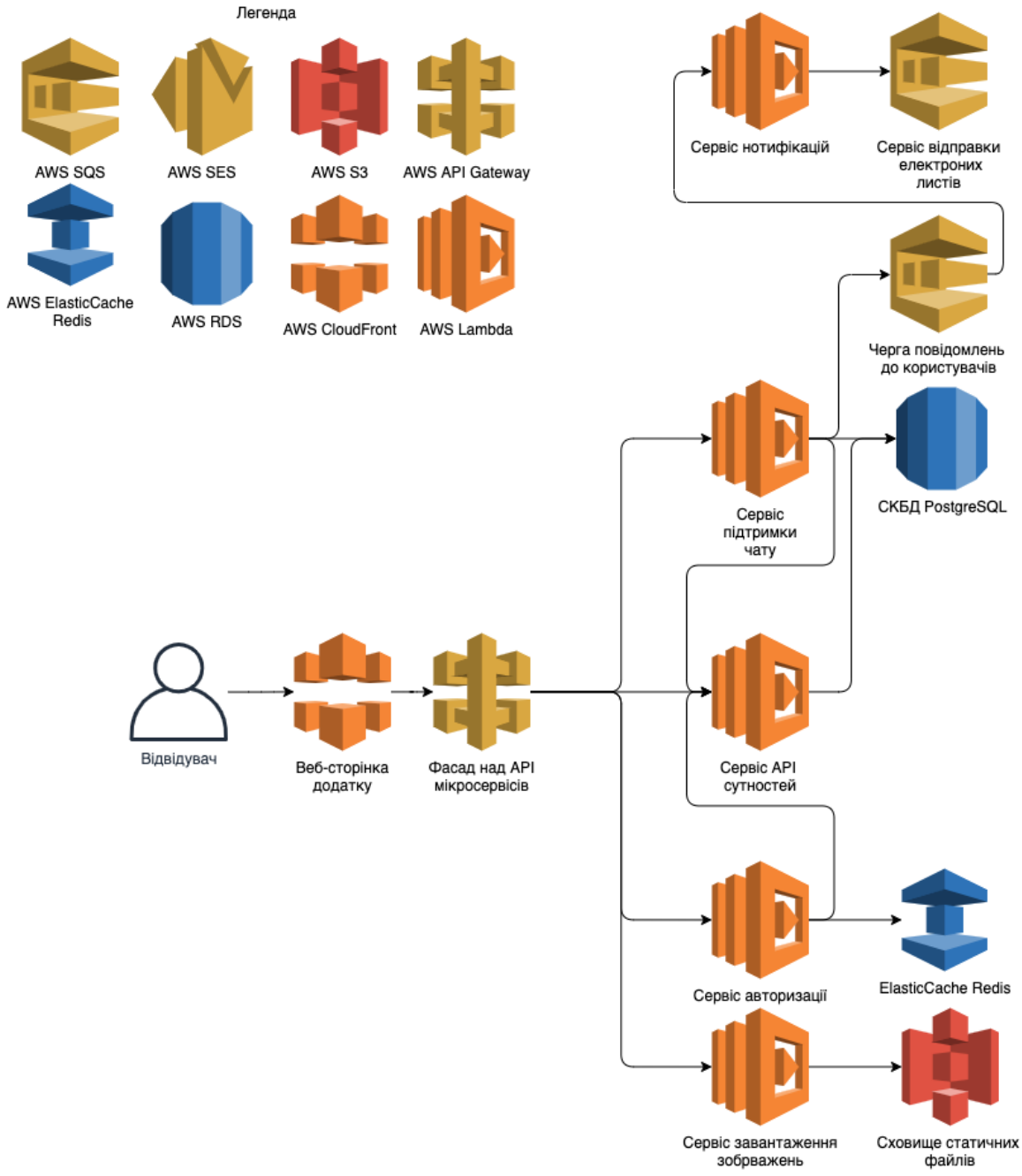


Рис. 3.10. Структура серверної частини веб-додатку в нотації архітектури AWS

Усі запити до серверу, які надходять від користувача, який користується додатком, оброблюються сервісом AWS API Gateway [50]. Він відповідає за представлення сервіс-орієнтованої архітектури у вигляді

монолітного API, з із єдиною точкою входу. Для того аби інтерфейс мікросервісу став доступний для використання у зовнішньому по відношенню до середовища додатку оточенні, усі його REST-шляхи мають бути описані у вигляді конфігурації AWS API Gateway. Це дозволяє на етапі отримання запиту виділяти із нього лише необхідну інформацію, наприклад параметри URL-адреси та IP адреса відправника запиту, тим самим зменшуючи навантаження на мережу. Кожен запит проходить через наступний цикл:

1. Отримання запиту від клієнта AWS API Gateway.
2. Визначення обробника запиту.
3. Якщо обробника немає, відіслати відповідь із помилкою. Завершити обробку, перейти до пункту 9.
4. Знайдення відповідної конфігурації, яка має бути застосована до даного запиту.
5. Обробка запиту, відокремлення необхідної інформації.
6. Формування запиту до обробника.
7. Відправка сформованого запиту, очікування відповіді.
8. Переадресування відповіді до клієнта-відправника.
9. Записати час відповіді.

Обробниками запитів від клієнтської частини веб-додатку виступають програми, запущені в оточенні сервісу AWS Lambda. Як зазначалося вище, вони працюють за принципом FaaS. Існує пул готових до використання екземплярів лямбда-функцій. Коли приходить запит, з цього пулу вилучається один обробник, який повертається туди після закінчення обробки запиту. Якщо вільних обробників не має, створюється новий. В разі, якщо певний час обробники не використовувалися, вони знищуються для економії ресурсів платформи AWS. Таким чином реалізовано горизонтальне масштабування системи – властивість системи справлятися із зростаючим навантаженням через збільшення екземплярів компонентів системи, замість залучення більших обчислювальних ресурсів для кожного

екземпляру.

Для авторизації запитів від клієнтів використовується мікросервіс авторизації. Він перевіряє надану користувачами інформацію та повертає статус авторизації.

За обробку запитів до сутностей у БД, зокрема до сутності Користувач, Повідомлення, Зображення, QR-код відповідає сервіс API сутностей. Він доступний до використання іншим сервісам, які хочуть створити або змінити записи у БД. Він приховує деталі реалізації зберігання даних програми від інших сервісів. Для цього він надає два види API – приватний та публічний. Публічний API підключено до AWS API Gateway та дозволено використовувати клієнтам, приватний API доступний тільки для прямого виклику, без використання AWS API Gateway.

Реалізацію операцій отримання даних про чати користувача та взаємодію із ними було виділено в окремий модуль – сервіс підтримки чату. Він дозволяє відмічати повідомлення як прочитані, отримувати відомості про кожен чат та кількість непрочитаних повідомлень у них, відправляти повідомлення та сповіщає отримувача про нові повідомлення. Для цього модуль додає задачу до черги AWS SQS [51].

Завдання із AWS SQS дістає споживач, який представляє собою сервіс нотифікацій, в зону відповідальності якого входить формування та відправка листів на електрону адресу користувачів. Безпосередня відправка листів реалізована із використанням AWS SES [52]. Це зручна система відправки повідомлень на електрону адресу, яку використовують маркетингові компанії.

Для обробки і збереження зображень у постійне сховище розроблено сервіс завантаження зображень. Він не перевіряє авторизацію користувача, очікуючи, що це зробили перед ним, а лише записує дані надісланих файлів після їх попередньої обробки в постійне сховище AWS S3 [53] – хмарний сервіс зберігання об’єктів.

Таким чином було реалізовано серверну частину веб-додатку для

сприяння поверненню втрачених речей. Як і зазначалося, основою для її побудови слугує оточення платформи хмарних обчислень AWS.

3.3. Структура БД

В результаті аналізу сформульованих вимог було спроєктовано наступні сутності, що зберігатимуться у базі даних системи.

3.3.1. *Користувач (User)*

- Ідентифікатор (ID) – унікальний ідентифікатор користувача у системі в форматі UUID.
- Ім’я (Name) – обране користувачем ім’я доступне усім користувачам системи, в тому числі не зареєстрованим.
- Улектронна пошта (Email) – адреса електронної скриньки користувача, застосовується при авторизації. Обов’язково має бути унікальною в рамках веб-додатку.
- Пароль (Password) – відповідно до вимоги SEC-2 буде зберігатися у захешованому вигляді за допомогою алгоритму SHA256 [X].
- Дата створення (Created at) – дата реєстрації користувача.
- Номер мобільного телефону (Phone) – не обов’язковий номер мобільного телефону, може бути наданий ОЯЗЧР в якості інформації про власника речі.
- Тип користувача (Type) – вказує на типу профілю користувача. Користувач може бути віртуальним, якщо ОЯЗЧВ не зареєстрована в системі та веде діалог із ВЗР, в іншому випадку тип користувача є реальним.
- Налаштування електронної адреси (Email preferences) – налаштування нотифікацій, які користувач бажає отримувати на електрону адресу.
- Флаг видалення (Deleted) – відповідає статусу профіля користувача. Оскільки повне видалення запису із БД про профіль

користувача є складною та комплексною операцією через велику кількість зв’язаних між собою сутностей, видалення відбувається через маркування потрібного запису як “видалений”.

3.3.2. *QR код (Qr code)*

- Ідентифікатор (ID) – унікальний ідентифікатор запису у таблиці.
- Назва (title) – назва QR-коду додана користувачем для спрощення його ідентифікаціїї.
- Хеш (hash) – строчка в якій зберігається поточний хеш QR-коду. Якщо користувач забажає деактивувати всі існуючі зображення із цим QR-кодом, ця строка буде перезаписана, в результаті чого посилання в QR-кодах стануть не валідні.
- Налаштування (Preferences) – налаштування візуального стилю зображення QR-коду.
- Ідентифікатор власника (User ID) – зовнішній ключ. Ідентифікатор запису про власника QR-коду.
- Флаг деактивації (Deactivated) – показує, чи був даний QR-код деактивований користувачем.

3.3.3. *Чат (Chat)*

- Ідентифікатор (ID) – унікальний ідентифікатор запису у таблиці.
- Ідентифікатор QR-коду (QR code ID) – зовнішній ключ, унікальний ідентифікатор запису про QR-код по якому звернулася ОЯЗЧВ до ВЗР.
- Дата останнього оновлення (Updated at) – дата останньої дії користувачів в чаті.
- Статус чату (status) – відображає поточний статус чату: не підтверджений, активний, деактивований.

3.3.4. *Повідомлення в чаті (Message)*

- Ідентифікатор (ID) – унікальний ідентифікатор повідомлення.
- Ідентифікатор відправника (Sender ID) – зовнішній ключ, ідентифікатор учасника чату, який його відправив.
- Ідентифікатор чату (Chat ID) – зовнішній ключ, посилання на дані про чат.
- Зміст (Content) – текст надісланого повідомлення.
- Дата створення (Created at) – дата надіслання повідомлення.
- Маркер, чи було повідомлення прочитане отримувачем (Was read) – після того як повідомлення відобразилося користувачу, воно автоматично помічається як прочитане.
- IP адреса відправника (Source IP) – IP адреса відправника, може бути використана для отримання геопозиції відправника.

3.3.5. *Учасник чату (Chat member)*

- Ідентифікатор (ID) – унікальний ідентифікатор учасника чату.
- Роль (Role) – роль яку має користувач у чаті.
- Ідентифікатор профіля користувача (User ID) – зовнішній ключ, посилання на профіль користувача, віртуальний чи реальний.
- Ідентифікатор чату (Chat ID) – зовнішній ключ, посилання на дані про чат.
- Кількість непрочитаних повідомлень у чаті (Unread count) – кількість нових повідомлень від іншої особи в чаті, які користувач ще не прочитав.
- Надана персональна інформація (Personal info) – відопості про персональні дані, які відображаються іншому учаснику чату.

3.3.6. *Запит на створення чату (Chat request)*

- Ідентифікатор (ID) – унікальний ідентифікатор запиту.

- Ідентифікатор чату (Chat ID) – зовнішній ключ, посилання на дані про чат.
- Дата створення (Created at) – дата створення запиту від ОЯЗЧВ.
- Прикріплені зображення (Attached photos URLs) – посилання на прикріплені до запиту зображення.
- IP адреса відправника (Source IP) – IP адреса відправника, може бути використана для отримання геопозиції відправника.
- Інформація про ОЯЗЧВ (Initiator info) – інформація яку надав ініціатор чату про себе при створенні запиту.

3.3.7. *ERD-діаграма БД*

Усі описані моделі сутностей в БД відповідають представленій схемі зв'язків (англ. "entity relations diagram") на рис. 3.11.

3.3.8. *Висновки*

В результаті проектування структури БД було отримано опис сутностей із якими оперує веб-додаток та їх властивостей, встановлено зв'язки між ними. На основі цих даних було розроблено запити до СУБД PostgreSQL для створення відповідної структури в БД веб-додатку.

3.4. Алгоритм автентифікація учасника чату

Розроблення веб-додатку для сприяння поверненню втрачених речей передбачає реалізацію чату, за допомогою якого ОЯЗЧР зможе зв'язатися із ВЗР. Цей чат дозволяє обом учасникам відправляти лише текстові повідомлення. При отриманні повідомлення, його адресант має отримувати нотифікацію на електрону адресу, за умови, що вона відома, або в особистий кабінет, якщо він зараз знаходиться на сайті.

Чат із ВЗР автоматично створюється після того, як ОЯЗЧР відправляє запит на ініціацію чату та прикріплює три фотографії, на яких зображено знайдену річ. Після цього, ОЯЗЧР потрапляє на сторінку із чатом, посилання

на яку виглядає як «https://<domain>/chat/<chat-token>», тут chat-token представлений у вигляді JWT [54], в ньому закодовано дані про ідентифікатор чату, в якому знаходяться ВЗР і ОЯЗЧР.

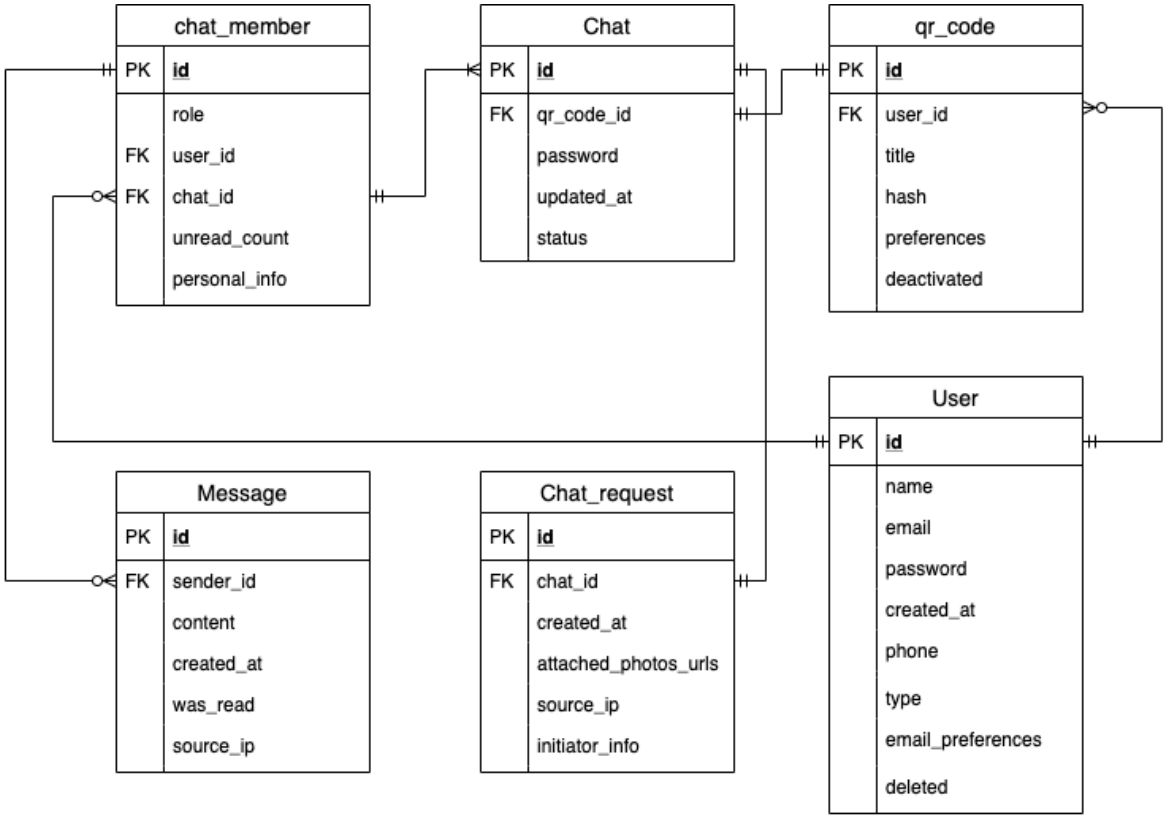


Рис. 3.11. ERD структура бази даних веб-додатку

Можливі два випадки, взаємодіє ОЯЗЧР із чатом:

- ОЯЗЧР автентифікована системою веб-додатку на момент ініціалізації чату: тоді вона може відкривати чат прямо із веб-інтерфейсу, в розділі «мої чати». Ніхто інший не зможе відкрити чат за згенерованим посиланням, оскільки всі його учасники відомі для системи. Такі чати називаються «закритими».
- ОЯЗЧР не автентифікована системою на момент створення нового чату: в такому разі, система вважає, що в чаті приймають участь ВЗР та анонімний користувач. Такі чати називаються «відкритими».

Розглянемо останній випадок більш детально.

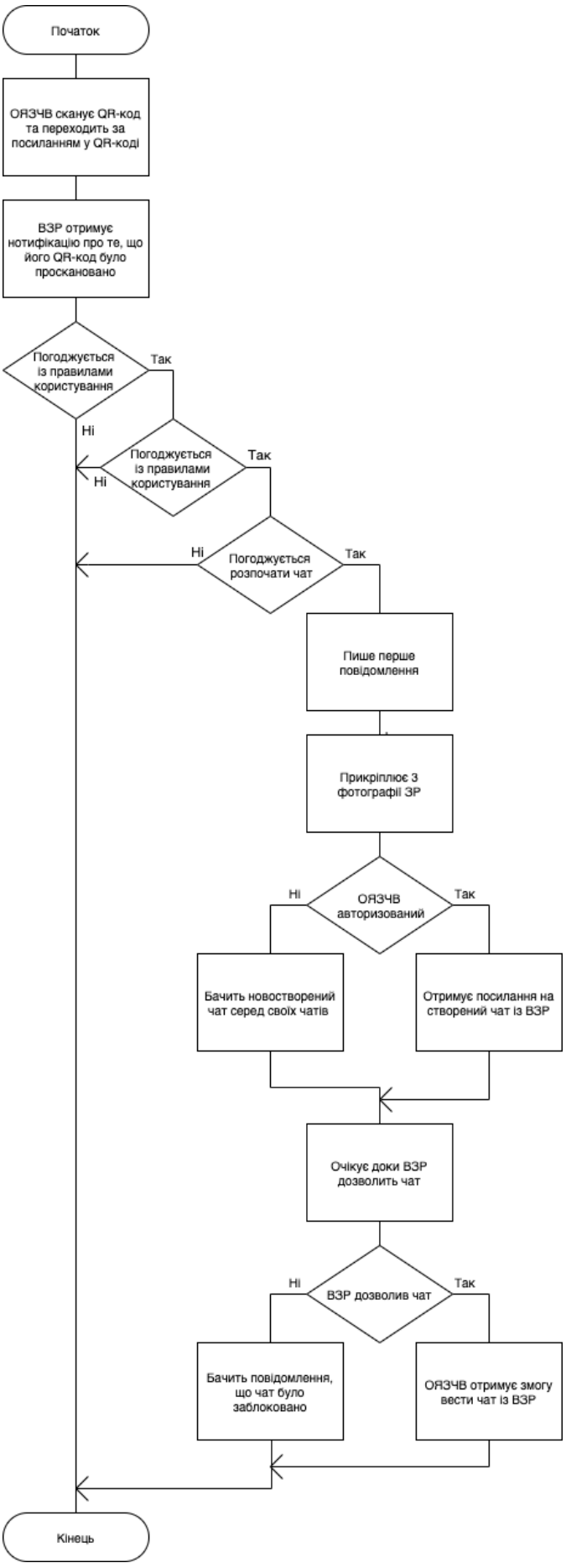


Рис. 3.12. Діаграма взаємодії. Дії ОЯЗЧВ для початку спілкування із ВЗР

Рис. 3.13. Схема процесу завантаження зображення на сервер

Одразу після створення заявки на ініціацію чату, системою буде запропоновано ОЯЗЧР зберегти посилання на чат для можливості повторно ввійти в нього вже після того, як попередня сесія завершиться. Анонімним користувачем, з точки зору веб-додатку може бути будь-який відвідувач, що відкрив у вікні браузеру посилання на чат. Усі спроби під’єднатися до існуючого чату за посиланням на нього потребуватимуть від ОЯЗЧР пройти

перевірку Google reCAPCHA [55]. Ця процедура гарантує, що користувач не є автоматичною системою. Після цього він має та надіслати посилання, яке закодоване в QR-коді на знайденій речі, для цього можна просто відправити фотографію, на якій зображено відсканований QR-код. Якщо пройти ці кроки, відвідувач ідентифікується системою як ОЯЗЧР, що ініціювала цей чат. Цей алгоритм представлено у виді діаграми взаємодії на рис. 3.12.

Окрім цього ОЯЗЧВ може авторизуватися пізніше та додати існуючий відкритий чат до свого профілю. В результаті чого чат автоматично стає закритим.

3.5. Алгоритм завантаження зображень у чаті

Функціональними вимогами до веб-додатку для сприяння поверненню втрачених речей передбачено реалізацію завантаження зображень на сервер. Алгоритм завантаження зображень представлено на рис. 3.13.

Загальний опис алгоритму:

1. Користувач додає зображення до форми, використовуючи drag’n’drop або файлову систему.
2. Система перевіряє, чи відповідає це зображення вимогам: мінімальна роздільна здатність 400 пікселів в ширину та 400 в висоту, максимальний розмір завантаженого файлу 5 МБ. Якщо зображення має один із вимірів більший, ніж 800 пікселів, система масштабує зображення таким чином, щоб його більша сторона стала 800 пікселів. При цьому інша сторона може стати меншою ніж 400 пікселів.
3. Надсилається POST запит до веб-серверу, із прикріпленим зображенням.
4. Веб-сервер перевіряє авторизацію користувача, що надіслав запит.

5. Веб-сервер завантажує зображення до хмарного сховища статичних файлів AWS S3.
6. Веб-сервер надсилає відповідь клієнту із посиланням на завантажене зображення.

4. ОСОБЛИВОСТІ РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1. Реалізація розгортання та тестування веб-додатку

Зі зростанням складності ПЗ та кількості його компонентів дедалі актуальнішою стає проблема налагодження процесу для розгортання коду в робочому оточенні. Робочим оточенням може слугувати виділений сервер, магазин мобільних додатків або, як у випадку розроблюваного веб-додатку, платформа хмарних обчислень AWS.

В IT налагодженням процесу доставки вихідного коду до середовища, де він буде працювати займаються спеціальні інженери. Методологія, за якою вони працюють називається DevOps – акронім від англійської developer та operations. Фактично такі спеціалісти об’єднують в своїй сфері діяльності практики розробки ПЗ та інформаційно-технологічного обслуговування систем. Завдання цих спеціалістів полягає в узгоджені та налагодженні розробки і постачання ПЗ. Це завдання вирішується за допомогою автоматичних засобів, побудованих за CI/CD (англ. «continuous integration / continuous delivery» – укр. «безперервна інтеграція/безперервна доставка») [56].

4.1.1. *Реалізація безперервної інтеграції коду*

Принципом безперервної інтеграції CI є розроблення вихідного коду малими ітераціями, кожна з яких має бути у короткі строки розгорнута на виробництві. Метою є виконання операцій створення, тестування і випуску надійніше і частіше. Одним із методів досягнення цього є описання послідовності дій, які мають бути виконані для тестування та розгортання застосунку у вигляді інструкцій. Цей процес має бути повторюваним та простим, настільки, щоб його могла виконати автоматична система. Прикладом цих систем є Gitlab CI, Jenkins, Circle CI [57]. Вони дозволяють описати послідовність дій та етапів, які слід виконати для того, аби робоча версія ПЗ була розгорнута.

В рамках розробки створюваного веб-додатку було використано

систему Gitlab CI [58] для створення конвеєру інтеграції змін у основну кодову базу Було реалізовано такі етапи перевірки коду:

1. Перевірка якості коду за допомогою інструменту eslint [59].
2. Запуск юніт-тестів за допомогою інструменту для тестування Jest [60].
3. Збір даних про тестування та покриття вихідного коду тестами.

Автоматизація процесу перевірки коду, перед його додавання до основної гілки кодової бази дозволило зменшити час, який потребується при оцінці цього злиття та пошуку помилок шляхом ручного запуску тестів в локальному оточенні.

4.1.2. *Реалізація безперервної доставки коду*

Після вдалого проходження кодом перевірки на якість, він зливався із основною гілкою кодової бази. В разі, якщо нова версія додатку була готова до розгортання в робочому оточенні, розробник ініціював початок процесу завантаження коду в AWS. Для цього було використано систему Gitlab CI. З її допомогою було описано конвеєр для розгортання коду.

Веб-додаток для сприяння поверненню втрачених речей для свого нормального функціонування потребує правильно налаштованої інфраструктури – ресурсів, які потрібні для підтримку ПЗ. До інфраструктури веб-додатку відносяться:

- сервіс СКБД AWS RDS;
- сервіс Черг завдань AWS SQS;
- сервіс відправки повідомлень електроною поштою AWS SES;
- сервіс менеджменту API AWS API Gateway;
- сервіс моніторингу AWS CloudWatch [61].

Контролювання версій та налаштувань цих сервісів є важким завданням для людини, через велику кількість змінних, яких потрібно враховувати при черговому оновленні версії додатку. Тому для виконання цієї роботи було використано підхід IaC.

IaC – це підхід до контролю і підтриманню архітектури ПЗ, через який її налаштування за допомогою декларативного опису в форматі конфігурації [62]. Проблемами, які вирішуються IaC є усунення необхідності людині виконувати рутині операції, мінімізація фактору людської помилки при оновленні інфраструктури, вирішення проблем дублювання та наслідування конфігурацій, спрощення процесу розгортання нової версій інфраструктури. Сучасні інструменти IaC дозволяють реалізувати автоматичний процес розгортання інфраструктури.

Популярним інструментом, для описання інфраструктури не тільки додатків на основі платформи AWS, а й інших постачальників хмарних обчислень є Terraform [63]. Terraform – ПЗ від компанії Hashicorp, яке дозволяє в форматі коду описати необхідні модулі та зв'язки між ними, які входять до архітектури ПЗ. Для оновлення розгорнутої інфраструктури, розробники повинні виконати наступні дії:

1. Описати інфраструктуру в форматі коду мовою Terraform.
2. Додати змінні оточення, ключі доступу до ресурсів платформи хмарних обчислень.
3. Запустити команду terraform plan, яка порівняє локально описану інфраструктури із завантаженою в хмару та обчислить різницю між ними.
4. Перевірити, що обчислена різниця між інфраструктурами коректна та не призведе до відмови всього ПЗ.
5. Виконати команду terraform apply, яка оновить інфраструктуру проєкту.

В рамках розроблюваного веб-додатку для сприяння поверненню втрачених речей користувача етапи 2-5 були реалізовані в системі Gitlab CI в рамках етапу "deploy" (укр. «розгортання»). Виконання найвідповідальнішого етапу 5 було зроблене мануальним, тобто таким, що потребує підтвердження дій від відповідального розробника, який зробить це через веб-інтерфейс Gitlab CI (рис. 4.1).

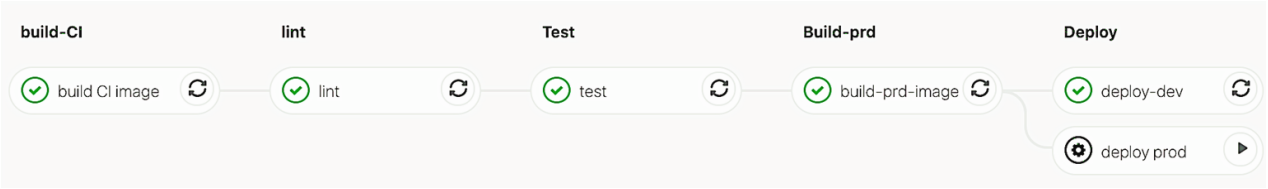


Рис. 4.1. Етапи конвеєру розгортання веб-додатку в системі Gitlab CI

4.2. Опис інтерфейсу користувача

Інтерфейс користувача розробленого веб-додатку було реалізовано у вигляді односторінкового (англ. “SPA”) веб-застосунку. Він складається із наступних сторінок:

1. Головна сторінка проєкту.
2. Домашня сторінка зареєстрованого користувача.
3. Сторінка перегляду чатів.
4. Сторінка чату ВЗР і ОЯЗЧВ.
5. Сторінка налаштувань профілю користувача.
6. Сторінка перегляду QR-кодів.
7. Сторінка створення та редагування QR-коду.
8. Сторінка створення запиту на початок спілкування ОЯЗЧВ із ВЗР.
9. Сторінка відновлення паролю.

При розробці вказаних сторінок використовувалися технології: веб-фреймворк React.js та бібліотека компонентів до нього SemanticUI [54].

Кольорова палітра веб-застосунку була обрана із розрахунку, що її кольори будуть викликати в людей відчуття спокою та довіри. Для цих цілей кольорова палітра створювалася із використанням синього та зеленого кольорі і їх відтінків. За результатами досліджень [65] синій колір викликає в людей відчуття спокою та захищеності. Зелений колір символізує позитивні тенденції, здоров’я та природу. Оскільки цільовою аудиторією цього веб-додатку є люди, які стикалися із втратою власності і хочуть її уникнути, важливо, аби вони всеціло довіряли цьому веб-додатку та покладалися на нього. З іншого боку, ОЯЗЧВ повинна отримати правильне

враження від інтерфейсу сторінки, де вона бачить інформацію про ВЗР. Цей інтерфейс не має викликати відчуття занепокоєння чи страху, что ОЯЧВ може бути звинувачений у крадіжці тощо. На думку авторів, обрана кольорова палітра, яка буде складатися із наступних кольорів якнайкраще підійде для цих цілей:

- «Snow» #F7F0F0;
- «Electric Blue» #8AF3FF;
- «Dark Liver» #484549;
- «Persian Green» #18A999;
- «Spanish Green» #109648.

Інші кольори були отримані як результат змішування цих основних, або як їх відтінки.

Кожна сторінка веб-додатку складається із трьох частин: шапки із меню зверху, контейнеру із контентом сторінки та підвалу із посиланнями на інформацію про проєкт та його розробників.

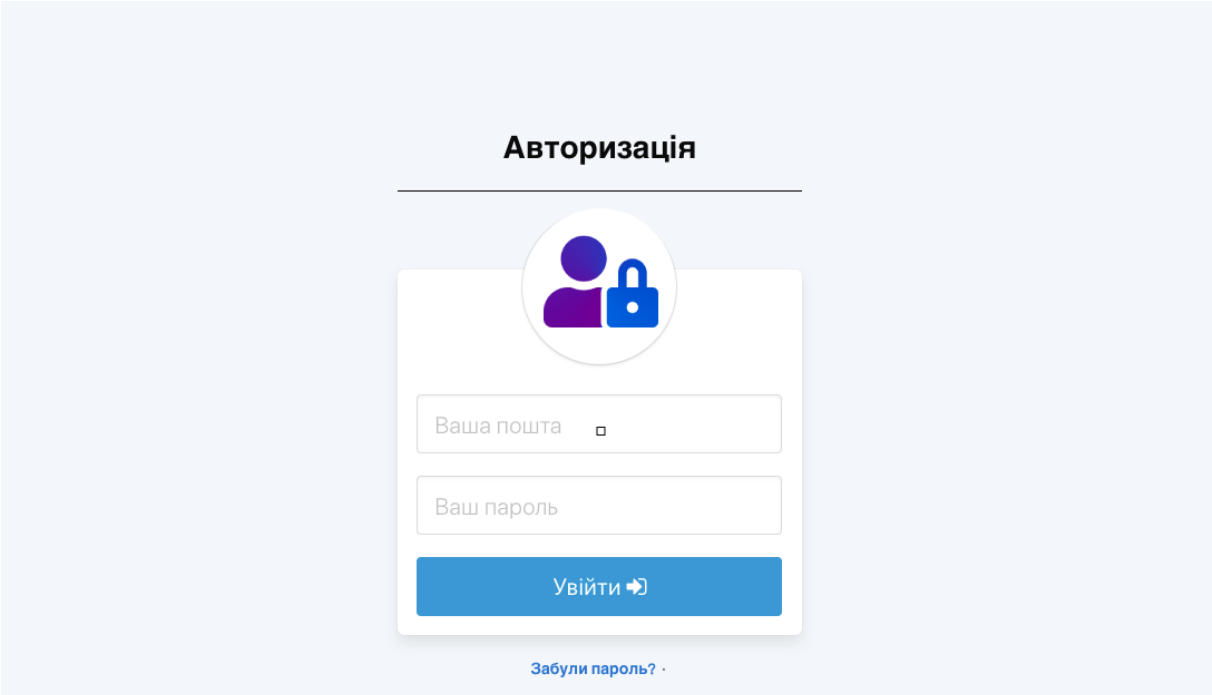


Рис. 4.2. Форма для авторизації користувача

Для того аби зробити додаток зручним і простим та зменшити

кількість дій, які користувачам слід виконати для досягнення їх мети, деякі функціональні модулі інтерфейсу було реалізовано у вигляді спливаючих вікон на сторінці. Наприклад, для того аби користувач міг авторизуватися із будь-якої веб-сторінки, форма для вводу електронної адреси та паролю, було реалізована таким чином (рис. 4.2).

Для створення нового чи редагування існуючого QR-коду було розроблено окрему сторінку із формою для вводу даних, які б користувач хотів зробити публічними та розмістити на сторінці, на яку посилається обраний QR-код. (рис. 4.3).

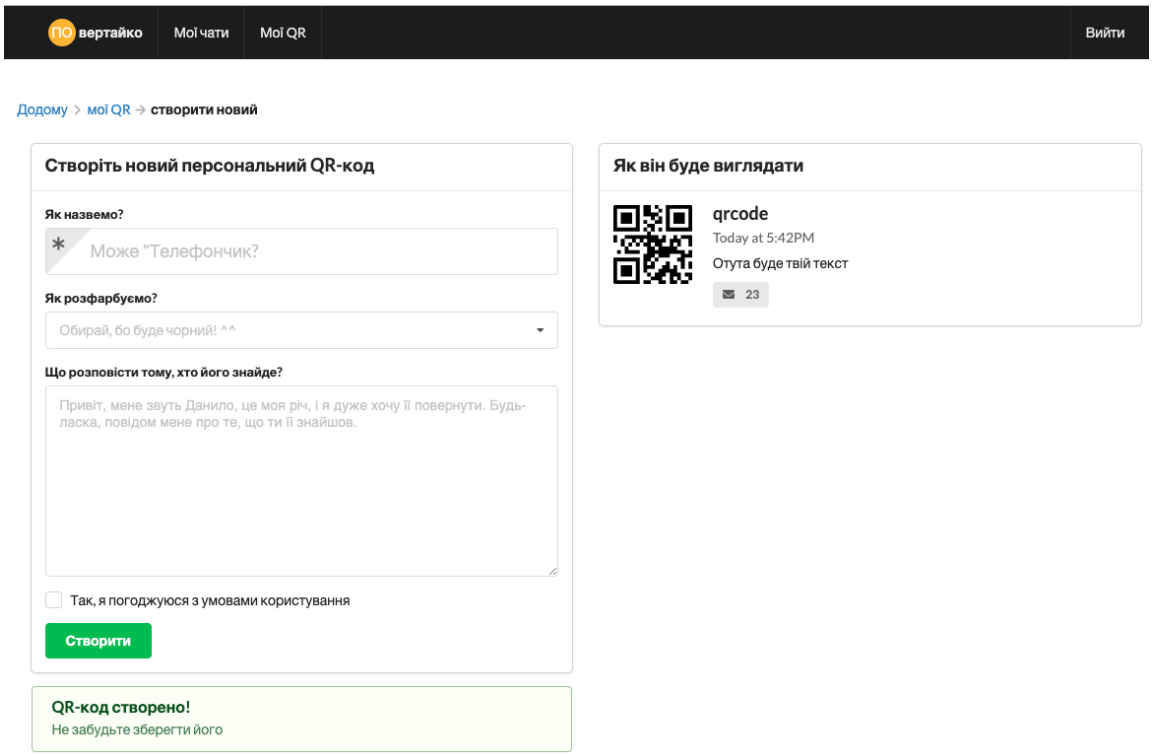


Рис. 4.3. Сторінка створення нового QR-коду

Оскільки кожен чат прив’язано до одного QR-коду, за яким звернулася ОЯЗЧВ, на сторінці перегляду всіх QR-кодів відображається інформація про кількість непрочитаних повідомлень у зв’язаних із кожним із них чатах, та посилання на них. Крім цього користувач має змогу завантажити QR-код за прямим посиланням, розміщеним на цій же сторінці. Приклад описаної сторінки знаходиться на рис. 4.4.

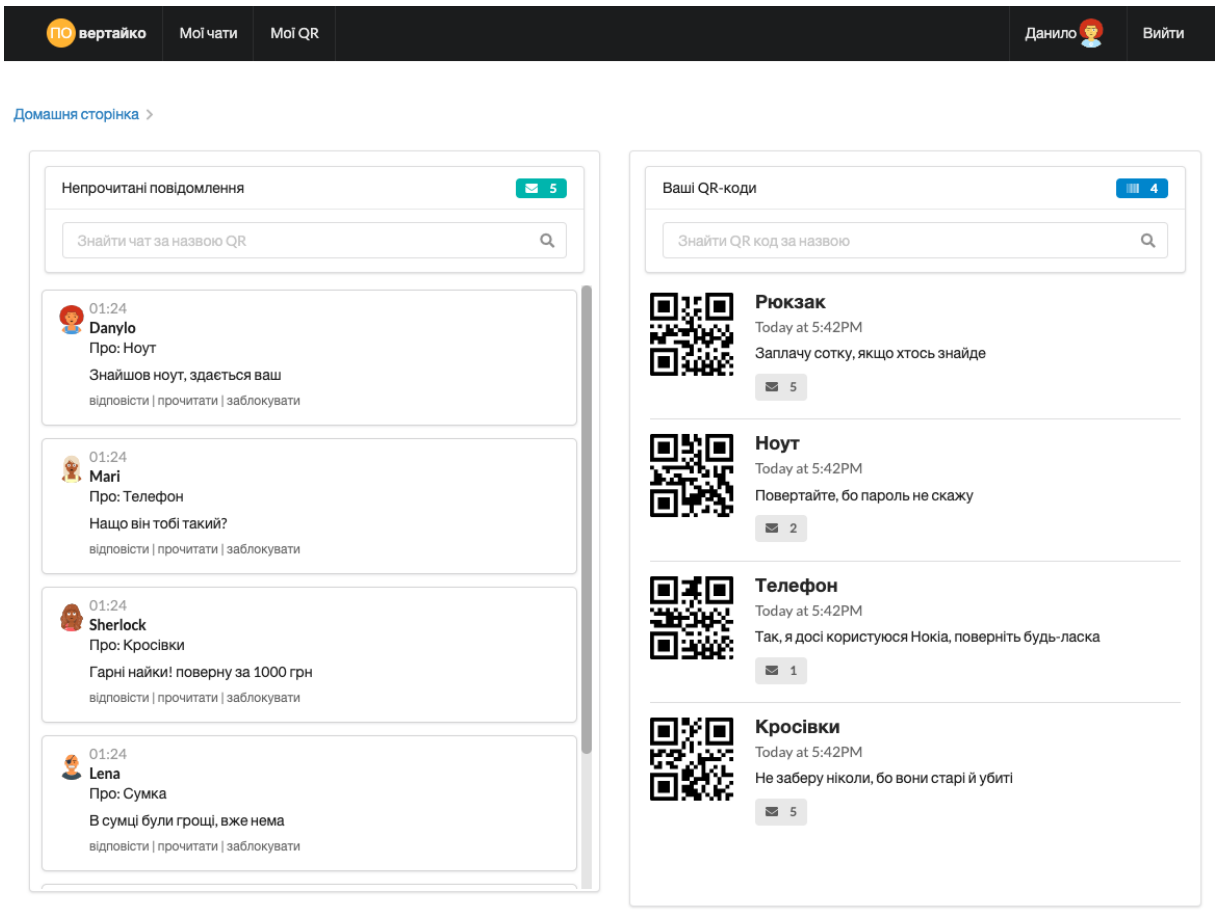


Рис. 4.4. Сторінка перегляду створених QR-кодів

Сторінка чату між користувачами містить інформацію про співрозмовника та про QR-код за яким звернулася ОЯЗЧВ. Кожне повідомлення складається із тексту, дати відправки та імені відправника. Повідомлення від поточного користувача вирівняні по правому краю, від його співрозмовника – по лівому. Це дозволяє швидко ідентифікувати автора повідомлення та є загальноприйнятним шаблоном проєктування інтерфейсів чатів. Приклад такої сторінки знаходить на рис. 4.5.

Важливу роль у веб-додатку відведено сторінці на яку ОЯЗЧВ перейде після сканування QR-коду. На цій сторінці розміщено інформацію, яка пояснює користувачу, що собою представляє веб-додаток. Крім цього на ній присутній блок із даними, які ВЗР надав про себе. Дизайн розроблено таким чином, щоб він закликав відвідувача до того, аби він написав ВЗР про свою знахідку та спробував її повернути. Аби досягти цього інтерфейс цієї сторінки було зроблено простим та вільним від зайвих елементів, які могли

б відволікати ОЯЗЧВ від тексту на ній. Фокус уваги відвідувача привертається до кнопки «написати», після натискання на яку він буде направлений на сторінку створення запиту на початок чату із ВЗР.

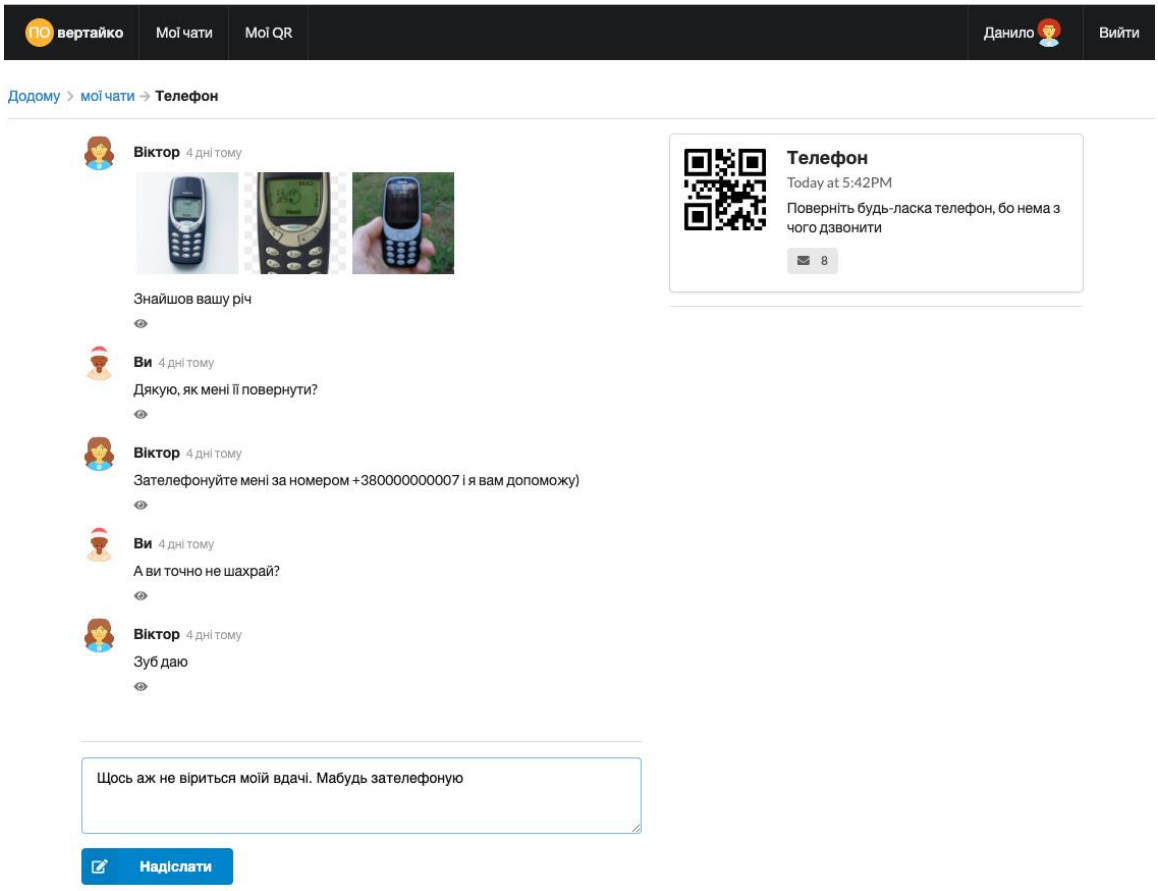


Рис. 4.5. Сторінка чату

4.3. Тестування веб-додатку

З ціллю підтвердити факт реалізації функціональних вимог, які висувались до веб-додатку для сприяння поверненню втрачених речей, було проведено тестування готового продукту. Для перевірки роботи застосунку було описано тестові випадки, які можна було застосувати в димовим (smoke) тестуванні.

В якості апаратної платформи, на якій відбувалося тестування було обрано ОС MacOs 10.15.14 із браузером Google Chrome 82. Тести перевіряються у наведеному нижче порядку. Результати попереднього тест-кейсу можуть впливати або використовуватися у подальших тест-

кейсах (табл. 4.1).

Таблиця 4.1

Тестові випадки димового тестування

№	Мета тест-кейсу	Дія	Очікуваний результат
1	Перевірити доступність веб-додатку за посиланням	Перейти за посиланням на веб-додаток у вікні браузеру	Відкривається головна сторінка веб-додатку
2	Перевірити можливість зареєструватися	1. Ввести дані у форму. 2. Натиснути на кнопку «зареєструватися».	Профіль користувача створено. В інтерфейсі стають доступні інші сторінки окрім головної.
3	Перевірити можливість увійти в профіль	1. Натиснути на кнопку «увійти». 2. Ввести дані у форму. 3. Натиснути на кнопку «увійти».	Користувач входить у свій профіль. В інтерфейсі стають доступні інші сторінки окрім головної.
4	Перевірити можливість відновити пароль	1. Натиснути на кнопку «увійти». 2. Ввести дані у форму. 3. Натиснути на кнопку «забув пароль». 4. Відкрити лист, що прийшов на вказану електрону адресу. 5. Перейти за посиланням в ньому. 6. Змінити пароль.	На електрону адресу прийде лист із посиланням на сторінку зміни паролю. Після зміни паролю, його можна використовувати для авторизації. Авторизація зі старим паролем вже не працює.
5	Перевірити вивід QR-кодів	Перейти на сторінку перегляду створених користувачем QR-кодів.	Відображається список створених поточним користувачем QR-кодів.

Продовження табл. 4.1

6	Перевірити можливість створити QR-код	1. Перейти на сторінку створення QR-коду. 2. Додати необхідну інформацію в форму. 3. Натиснути «зберегти».	Новий QR-код відображається в списку створених поточним користувачем QR-кодів, якщо його відсканувати, відкриється сторінка створення чату.
7	Перевірити можливість деактивувати QR-код	1. Перейти на сторінку перегляду створених користувачем QR-кодів. 2. Деактивувати один із активних QR-кодів.	QR-код не відображається в списку всіх QR-кодів. При переході за посиланням, яке було в нього закодоване не відображається стара сторінка інформації про користувача.
8	Перевірити можливість ОЯЗЧВ створити запит на початок чату із ВЗР	1. Бути не зареєстрованим користувачем. 2. Перейти за посиланням в QR- коді. 3. Додати фотографії знайденої речі. 4. Створити запит на ініціацію чату. 5. Додати фотографії знайденої речі.	Запит на створення чату прийшов ВЗР.
9	Перевірити можливість ВЗР дозволити чат	1. Бути не зареєстрованим користувачем. 2. Перейти за посиланням в QR- коді. 3. Додати фотографії знайденої речі. 4. Створити запит на ініціацію чату	ВЗР отримує на пошту оповіщення, про запит на початок чату. ОЯЗЧВ отримує посилання, за яким відкривається сторінка із створеним чатом.

Продовження табл. 4.1

10	Перевірити можливість ВЗР заборонити чат із ОЯЗЧВ	1. Авторизуватися як ВЗР, якому прийшло оповіщення про ініціацію чату від ОЯЗЧР. 2. Заборонити чат.	Чат не відображається у списку всіх чатів ВЗР. ОЯЗЧВ бачить в чаті, що чат був заблокований.
11	Перевірити можливість ВЗР дозволити чат із ОЯЗЧВ	1. Авторизуватися як ВЗР. 2. Перейти на сторінку чатів. 3. Знайти новий запит на створення чату. 4. Дозволити чат.	Чат відображається у списку всіх чатів ВЗР. ОЯЗЧВ бачить в чаті, що чат був дозволений.
12	Перевірити можливість ВЗР написати повідомлення в чаті	1. Авторизуватися як ВЗР. 2. Дозволити чат із ОЯЗЧВ. 3. Написати повідомлення в чаті.	Повідомлення зберігається в історії повідомлень чату. ОЯЗЧВ бачить нове повідомлення відправлене від ВЗР. Це повідомлення також бачить і ВЗР.
13	Перевірити можливість ОЯЗЧВ написати повідомлення в чаті	1. Бути не авторизованим. 2. Відкрити чат за посиланням. 3. Написати повідомлення.	Повідомлення зберігається в історії повідомлень чату. ВЗР бачить нове повідомлення.
14	Перевірити можливість ОЯЗЧВ продивитися інформацію про ВЗР на сторінці на яку вело посилання у QR-коді	1. Бути не авторизованим. 2. Перейти за посиланням в QR-коді. 3. Продивитися інформацію, яку ВЗР додав про себе.	Відображається коректна публічна інформація про ВЗР.

Продовження табл. 4.1

15	Перевірити можливість ОЯЗЧВ увійти в свій профіль та додати відкритий чат	1. Створити відкритий чат. 2. Авторизуватися як ОЯЗЧВ. 3. Додати посилання на чат.	Чат стає закритим та не відкривається за прямим посиланням, якщо користувач не авторизований.

4.4. Рекомендації щодо подальшого вдосконалення

В результаті реалізації описаного веб-додатку для сприяння поверненню втрачених речей на основі платформи AWS було знайдено шляхи для подальшого вдосконалення даного продукту.

До них відносяться:

- розширення функціональних можливостей з персоналізації QR коду: вибір кольору, форми, додавання власного логотипу, тощо.
- інтеграція із сторонніми сервісами авторизації за протоколом OAtuh2 [66].;
- додавання можливості надсилати в чаті зображення учасниками.
- виведення статистики для ВЗР про кожне сканування їх QR-кодів;
- використання Push-нотифікацій як альтернативу сповіщенням на електрону адресу;
- отримання даних про геопозиції ОЯЗЧВ за його IP-адресою та надавати ці дані ВЗР.

ВИСНОВКИ

Метою даного дипломного проєкту було вирішення проблеми низького відсотку повернених речей їх власникам за рахунок розроблення веб-додатоку для сприяння поверненню втрачених речей на основі платформи AWS. Результатом роботи став програмний продукт, який було реалізовано у вигляді веб-сайту, доступного до використання користувачам мережі Інтернет.

Після проведення дослідження проблем, з якими стикається цільова аудиторія розробленого додатку та аналізу існуючих аналогів, було встановлено, що проєктування та розроблення додатку для сприяння поверненню втрачених речей є доцільним та актуальним на сьогоднішній день. Було проаналізовано представлені на ринку технології для розроблення ПЗ. В результаті чого було обрано мову програмування TypeScript та веб-фреймворк React.js для розроблення клієнтської частини, мову програмування TypeScript на основі платформи Node.js для реалізації серверної частини, в якості СУБД було обрано PostgreSQL. Програмний комплекс було розроблено із використанням принципів сервіс-орієнтованої архітектури, що дозволило розгорнути веб-додаток в середовищі платформи хмарних обчислень AWS із використанням FaaS AWS Lambda. Було налагоджено процеси автоматичного CI/CD.

Розроблення виконано в повному обсязі, воно відповідає висунутим до нього вимогам, які описані в цьому документі, тестування продукту виконано у відповідності до затвердженої програми та методики тестування.

Використання даного веб-додатку дозволить підвищити відсоток повернутих втрачених речей їх господарям за рахунок вирішення основних проблем, які зазвичай заважали цьому.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. The 2019 Uber Lost & Found Index [Електронний ресурс]. — Режим доступу: https://www.uber.com/newsroom/2019-lost-and-found-index/
2. Virtual Lost and Found [Електронний ресурс]. — Режим доступу: https://www.computersciencezone.org/virtual-lost-found/
3. Why Japan is so successful at returning lost property [Електронний ресурс]. — Режим доступу: https://www.bbc.com/future/article/20200114-why-japan-is-so-successful-at-returning-lost-property/
4. QR-код [Електронний ресурс]. — Режим доступу: https://ru.wikipedia.org/wiki/QR-код
5. Find My [Електронний ресурс]. — Режим доступу: https://www.apple.com/icloud/find-my/
6. SAS [Електронний ресурс]. — Режим доступу: https://www.flysas.com/ru-ru/
7. Paritet bank [Електронний ресурс]. — Режим доступу: https://www.paritetbank.by/private/
8. Донаван, А. Язык программирования Go [Текст] / А. Донаван, Б. Керниган. — СПб. : ООО «И. Д. Вильямс», 2016. — 432 с.
9. Documentation of Go [Електронний ресурс]. — Режим доступу: https://golang.org/doc/
10. Лутц, М. Изучаем Python 4-е издание [Текст] / М. Лутц. — СПб. : ООО «И. Д. Вильямс», 2011. — 1280 с.
11. What is Python: [Електронний ресурс]. — Режим доступу: https://www.pythonforbeginners.com/learn-python/what-is-python/
12. Mypy [Електронний ресурс]. — Режим доступу: http://mypy-lang.org/
13. Asyncio: [Електронний ресурс]. — Режим доступу: https://www.pythonforbeginners.com/learn-python/what-is-python/

14. Python Performance Tips [Електронний ресурс]. — Режим доступу: https://wiki.python.org/moin/PythonSpeed/PerformanceTips
15. What is Java [Електронний ресурс]. — Режим доступу: https://www.edureka.co/blog/what-is-java/
16. Паттерны проєктирования [Електронний ресурс]. — Режим доступу: https://refactoring.guru/ru/design-patterns/java
17. JavaScripts basics [Електронний ресурс]. — Режим доступу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/JavaScript_basics
18. Прототипне програмування [Електронний ресурс]. — Режим доступу: https://uk.wikipedia.org/wiki/Прототипне_програмування
19. Node.js [Електронний ресурс]. — Режим доступу: https://nodejs.org
20. How JavaScript works: inside the V8 engine + 5 tips on how to write optimized code [Електронний ресурс]. — Режим доступу: https://blog.sessionstack.com/how-javascript-works-inside-the-v8-engine-5-tips-on-how-to-write-optimized-code-ac089e62b12e
21. Python vs NodeJS which better your project [Електронний ресурс]. — Режим доступу: https://da-14.com/blog/python-vs-nodejs-which-better-your-project
22. The State of Developer Ecosystem 2019 [Електронний ресурс]. — Режим доступу: https://www.jetbrains.com/lp/devecosystem-2019/
23. NPM | build amazing things [Електронний ресурс]. — Режим доступу: https://npmjs.com
24. TypeScript – JavaScript that scales [Електронний ресурс]. — Режим доступу: https://typescriptlang.org
25. How JavaScript became the dominant language of the web [Електронний ресурс]. — Режим доступу: https://www.lform.com/blog/post/how-javascript-became-the-dominant-language-of-the-web
26. AJAX – Руководско Web-разработчика [Електронний ресурс]. — Режим доступу: https:// developer.mozilla.org/ru/docs/Web/Guide/AJAX

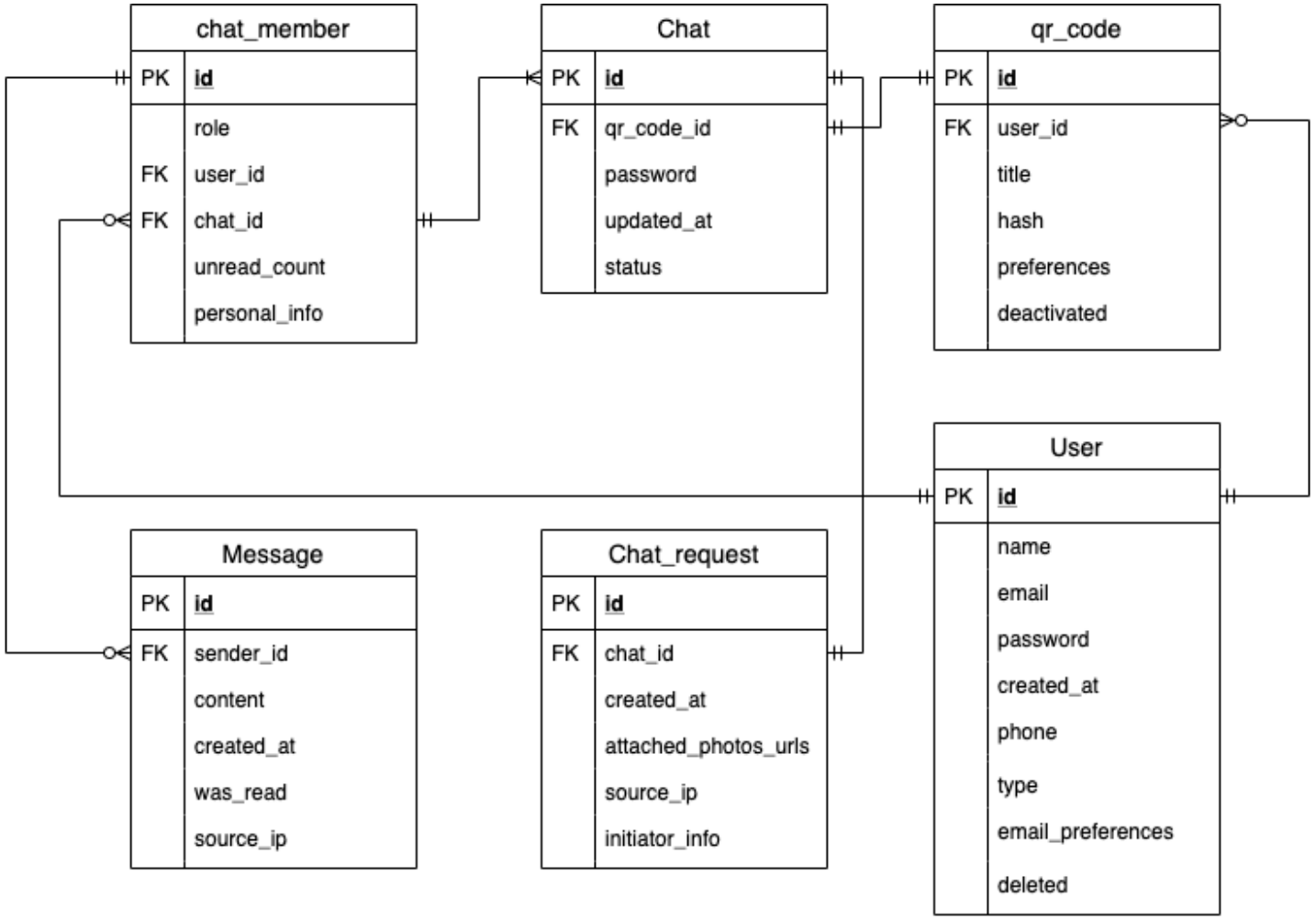
27. Angular vs React vs Vue a performance comparision [Електронний ресурс]. — Режим доступу: https://blog.logrocket.com/angular-vs-react-vs-vue-a-performance-comparison/
28. Vue.js Guide [Електронний ресурс]. — Режим доступу: https://vuejs.org/v2/guide/
29. Angular vs Vue vs React [Електронний ресурс]. — Режим доступу: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/
30. Angular vs. React vs. Vue.js – choosing a JavaScript framework for your project [Електронний ресурс]. — Режим доступу: https://relevant.software/blog/angular-vs-react-vs-vue-js-choosing-a-javascript-framework-for-your-project/
31. Angular [Електронний ресурс]. — Режим доступу: https://angular.io
32. React [Електронний ресурс]. — Режим доступу: https://ru.reactjs.org
33. Developer Survey Results [Електронний ресурс]. — Режим доступу: https://insights.stackoverflow.com/survey/2019
34. React hooks Overview [Електронний ресурс]. — Режим доступу: https://ru.reactjs.org/docs/hooks-overview.html
35. Система управления базами данных [Електронний ресурс]. — Режим доступу: https://ru.wikipedia.org/wiki/Система_управления_базами_данных
36. SQL [Електронний ресурс]. — Режим доступу: https://ru.wikipedia.org/wiki/SQL
37. NoSQL [Електронний ресурс]. — Режим доступу: https://ru.wikipedia.org/wiki/NoSQL
38. SQL or NoSQL that is the question [Електронний ресурс]. — Режим доступу: https://blog.panoply.io/sql-or-nosql-that-is-the-question
39. Redis [Електронний ресурс]. — Режим доступу: https://redis.io
40. AWS RDS [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/rds/
41. MySQL [Електронний ресурс]. — Режим доступу: https://mysql.com

42. PostgreSQL vs MySQL [Електронний ресурс]. — Режим доступу: https://www.postgresqltutorial.com/postgresql-vs-mysql/
43. PostgreSQL [Електронний ресурс]. — Режим доступу: https://www.postgresql.org
44. Function as a service [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Function_as_a_service
45. Cloud Functions [Електронний ресурс]. — Режим доступу: https://cloud.google.com/functions
46. Azure Functions [Електронний ресурс]. — Режим доступу: https://azure.microsoft.com/en-us/services/functions/
47. AWS Lambda [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/lambda/
48. IBM Developer. Сервис, архитектура, управление и бизнес-термины [Електронний ресурс]. — Режим доступу: https://www.ibm.com/developerworks/ru/library/ws-soa-term1/
49. AWS CloudFront [Електронний ресурс]. — Режим доступу:https://aws.amazon.com/ru/cloudfront/
50. AWS ApiGateway [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/api-gateway/
51. AWS SQS [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/sqs/
52. AWS SQS [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/ses/
53. AWS S3 [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/s3/
54. JSON Web Token [Електронний ресурс]. — Режим доступу: https://jwt.io
55. Google ReCAPTCHA [Електронний ресурс]. — Режим доступу: https://www.google.com/recaptcha/intro/v3.html
56. Continuous integration vs. continuous delivery vs. continuous deployment [Електронний ресурс]. — Режим

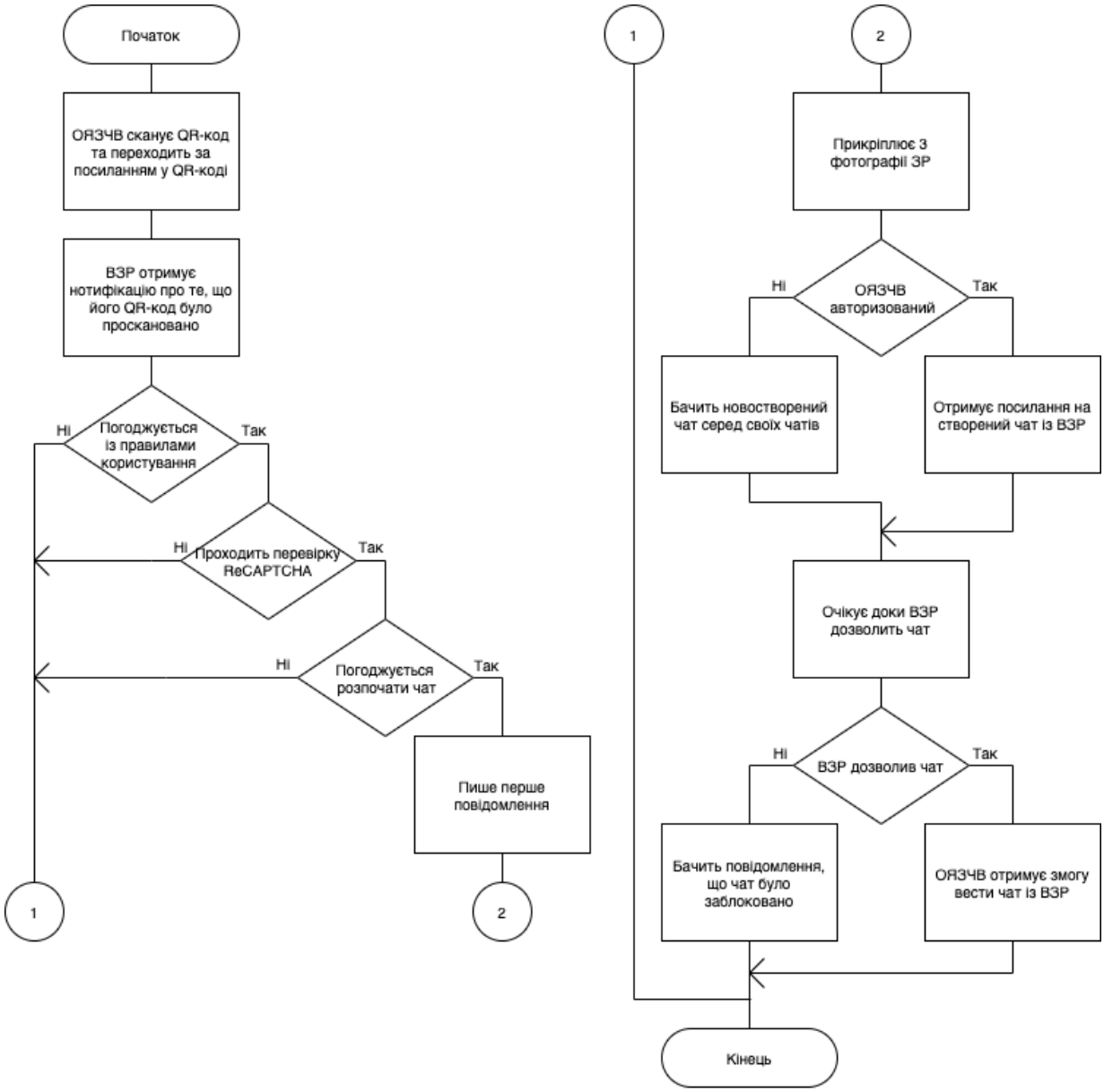
доступу: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment
57. Continuous Integration Essentials [Електронний ресурс]. — Режим доступу: https://codeship.com/continuous-integration-essentials
58. GitLab CI/CD [Електронний ресурс]. — Режим доступу: https://docs.gitlab.com/ee/ci/
59. ESlint – Pluggable JavaScript Linter [Електронний ресурс]. — Режим доступу: https://eslint.org
60. Jest – Delightful JavaScript Testing [Електронний ресурс]. — Режим доступу: https://jestjs.io
61. AWS CloudWatch [Електронний ресурс]. — Режим доступу: https://aws.amazon.com/ru/cloudwatch/
62. Infrastructure as code [Електронний ресурс]. — Режим доступу: https://en.wikipedia.org/wiki/Infrastructure_as_code
63. Terraform by Hashicorp [Електронний ресурс]. — Режим доступу: https://terraform.io
64. SemanticUi by Hashicorp [Електронний ресурс]. — Режим доступу: https:/semantic-ui.com
65. 12 Essential Tips to Picking a Website Color Scheme [Електронний ресурс]. — Режим доступу: https://neilpatel.com/blog/website-color-scheme/
66. OAuth 2.0 [Електронний ресурс]. — Режим доступу: https://oauth.net/2/

ДОДАТКИ

Додаток 1
Копії графічних матеріалів

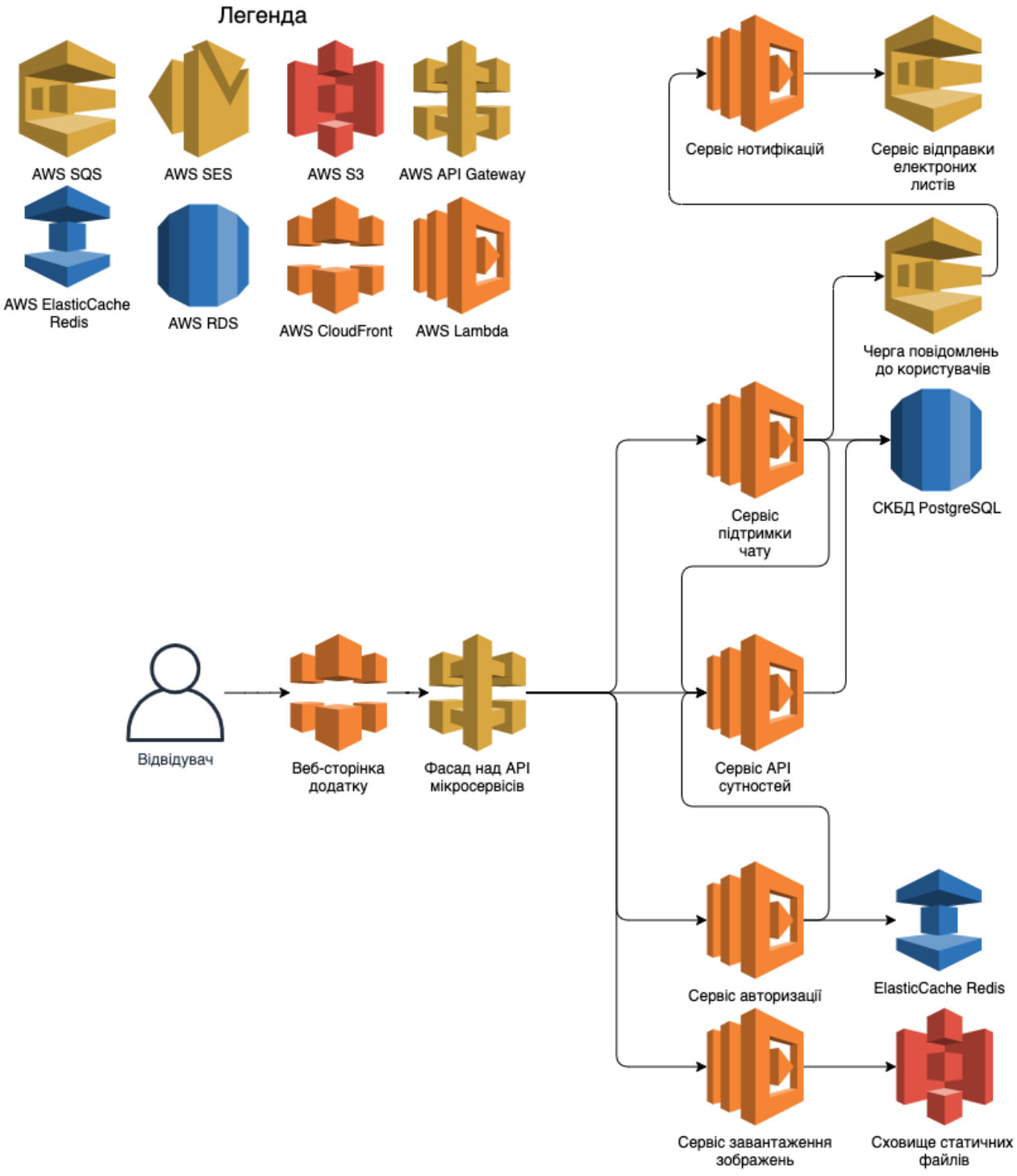


ДП.045440-06-99. Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS. Структура бази даних. ERD-діаграма



ДП.045440-07-99. Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS. Взаємодія особи, яка знайшла чужу річ, із веб-додатком. Діаграма діяльності

СХЕМА СТРУКТУРИ СЕРВРНОЇ ЧАСТИНИ ВЕБ-ДОДАТКУ ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS



Казимиров Д.М., група КП-61

ДЕРЕВО ПРОБЛЕМ ВЕБ-ДОДАТКУ ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS

Казимиров Д.М., група КП-61

Додаток 2
Лістинг програми

```
const Router = require('koa-router');
const { middleware: validation } = require('@zulus/request-validator');
const registerHandler = require('../handlers/auth/register');
const loginHandler = require('../handlers/auth/login');
const getAccessToken = require('../handlers/auth/getAccessToken');
const { authenticate } = require('../middlewares/auth');

const router = Router();

router.post('/register', validation(registerHandler.schema),
registerHandler);
router.post('/login', authenticate('local'), loginHandler);
router.get('/token', authenticate('jwt-refresh'), getAccessToken);
router.get('/google', authenticate('google-token'), registerHandler);

module.exports = router;

const { Model } = require('objection');
const constants = require('../constants');
const helpers = require('../helpers');
const { pick } = require('lodash');
const uuid = require('uuid/v4');

class User extends Model {
  static get idColumn() {
    return 'uid';
  }

  static get tableName() {
    return 'users';
  }

  static get jsonSchema() {
    return {
      type: 'object',
      required: [
        'email',
        'password'
      ],
      properties: {
        uid: { type: 'string' },
        password: { type: 'string' },
        role: { type: 'string' },
        createdAt: { type: 'string' },
        updatedAt: { type: 'string' }
      }
    };
  }

  async $beforeInsert(queryContext) {
    this[User.idColumn] = uuid();
    this.password = await helpers.passwords.encryptPassword(this.password,
constants.passwordSaltRounds);
    await super.$beforeInsert(queryContext);
  }

  async $beforeUpdate(queryContext) {
    this.password = await helpers.passwords.encryptPassword(this.password,
constants.passwordSaltRounds);
    await super.$beforeUpdate(queryContext);
  }

  static async isEmailTaken(email, { trx } = {}) {
    const result = await User.query(trx)
```

```
      .where({ email })
      .count();
    return !result[0].count;
  }

  static findByEmail(email) {
    return User.query()
      .findOne({ email });
  }

  toPrivateJson() {
    return pick(this, ['uid', 'email', 'createdAt', 'updatedAt']);
  }

  toPublicJson() {
    return pick(this, ['uid', 'createdAt', 'updatedAt']);
  }
}

module.exports = User;

const statusCodes = require('http-status');

const clearEmptyBodyFields = (ctx) => {
  if (ctx.body) {
    Object.keys(ctx.body)
      .forEach(key => {
        if (ctx.body[key] === null || ctx.body[key] === undefined) {
          delete ctx.body[key];
        }
      });
  }
};

module.exports = () => async (ctx, next) => {
  ctx.res.statusCodes = statusCodes;
  ctx.statusCodes = ctx.res.statusCodes;

  ctx.res.success = ({ statusCode, data = null, message = null } = {}) => {
    const status = 'ok';
    if (!!statusCode && (statusCode < 400)) {
      ctx.status = statusCode;
    } else if (!(ctx.status < 400)) {
      ctx.status = statusCodes.OK;
    }
    ctx.body = {
      status,
      data,
      message,
      statusCode
    };
  };
  ctx.res.ok = ctx.res.success;
  ctx.res.error = ({ statusCode, code, data = null, message = null } = {})
=> {
    const status = 'error';
    if (!!statusCode && (statusCode >= 400 && statusCode < 600)) {
      ctx.status = statusCode;
    } else if (!(ctx.status >= 500 && ctx.status < 600)) {
      ctx.status = statusCodes.INTERNAL_SERVER_ERROR;
    }
    if (!code) {
      code = statusCodes[`${ctx.status}_NAME`];
    }
```

```
  ctx.body = {
    status,
    code,
    data,
    message,
    statusCode
  };
};
ctx.res.accepted = (params = {}) => {
  ctx.res.success({
    ...params,
    statusCode: statusCodes.ACCEPTED
  });
};

ctx.res.noContent = (params = {}) => {
  ctx.res.success({
    ...params,
    statusCode: statusCodes.NO_CONTENT
  });
};

ctx.res.badRequest = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.BAD_REQUEST
  });
};

ctx.res.unauthorized = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.UNAUTHORIZED
  });
};

ctx.res.forbidden = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.FORBIDDEN
  });
};

ctx.res.notFound = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.NOT_FOUND
  });
};

ctx.res.requestTimeout = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.REQUEST_TIMEOUT
  });
};

ctx.res.unprocessableEntity = (params = {}) => {
  ctx.res.error({
    ...params,
    statusCode: statusCodes.UNPROCESSABLE_ENTITY
  });
};
```

```
  ctx.res.internalServerError = (params = {}) => {
    ctx.res.error({
      ...params,
      statusCode: statusCodes.INTERNAL_SERVER_ERROR
    });
  };

  ctx.res.notImplemented = (params = {}) => {
    ctx.res.error({
      ...params,
      statusCode: statusCodes.NOT_IMPLEMENTED
    });
  };

  ctx.res.badGateway = (params = {}) => {
    ctx.res.error({
      ...params,
      statusCode: statusCodes.BAD_GATEWAY
    });
  };

  ctx.res.serviceUnavailable = (params = {}) => {
    ctx.res.error({
      ...params,
      statusCode: statusCodes.SERVICE_UNAVAILABLE
    });
  };

  ctx.res.gatewayTimeOut = (params = {}) => {
    ctx.res.error({
      ...params,
      statusCode: statusCodes.GATEWAY_TIMEOUT
    });
  };

  await next();
  clearEmptyBodyFields(ctx);
};

const { Model } = require('objection');
const { pick } = require('lodash');

module.exports = class Messages extends Model {
  static get tableName() {
    return 'messages';
  }

  static get idColumn() {
    return 'id';
  }

  static get jsonSchema() {
    return {
      type: 'object',
      required: ['chatUid', 'message'],
      properties: {
        id: { type: 'number' },
        chatUid: { type: 'string' },
        userUid: { type: 'string' },
        message: { type: 'string' },
        createdAt: { type: 'string' }
      }
    };
  }
```

```
  static get relationMappings() {
    return {
      chat: {
        relation: Model.BelongsToOneRelation,
        join: {
          from: 'messages.chat_uid',
          to: 'chats.uid'
        },
        modelClass: `${__dirname}/Chats`
      },
      user: {
        relation: Model.BelongsToOneRelation,
        join: {
          from: 'messages.user_uid',
          to: 'users.uid'
        },
        modelClass: `${__dirname}/User`
      }
    };
  }
  static create(data, { trx } = {}) {
    return this.query(trx)
      .insert(data)
      .returning('*');
  }
  toPublicJson() {
    return pick(this, ['createdAt', 'message', 'userUid', 'id']);
  }
  static getByChatUid(chatUid, { trx, sortBy, sortDir, skip, limit } = {
    sortBy: 'createdAt',
    sortDir: 'desc',
    skip: 0,
    limit: Infinity
  }) {
    return this.query(trx)
      .where({ chatUid })
      .orderBy(sortBy, sortDir)
      .offset(skip)
      .limit(limit);
  }
};
const { Model } = require('objection');
const { pick } = require('lodash');

module.exports = class PendingThings extends Model {
    static get standartEagerOpts() {
        return '';
    }

    static get deepEagerOpts() {
        return '[thing, chat]';
    }

    static get tableName() {
        return 'pending_things';
    }

    static get idColumn() {
        return 'id';
    }

    static get jsonSchema() {
        return {
```

```
            type: 'object',
            required: ['charUid', 'pictureUrl'],
            properties: {
                id: { type: 'number' },
                chatUid: { type: 'string' },
                thingId: { type: 'string' },
                status: { type: 'string' },
                pictureUrl: { type: 'string' },
                createdAt: { type: 'string' },
                updatedAt: { type: 'string' }
            }
        };
    }

    static get relationMappings() {
        return {
            chat: {
                relation: Model.BelongsToOneRelation,
                join: {
                    from: 'pending_things.chatUid',
                    to: 'chats.uid'
                },
                modelClass: `${__dirname}/Chats`
            },
            thing: {
                relation: Model.BelongsToOneRelation,
                join: {
                    from: 'pending_things.thingId',
                    to: 'things.id'
                },
                modelClass: `${__dirname}/Things`
            }
        };
    }

    static create(data, { trx } = {}) {
        return this.query(trx)
            .insert(data)
            .returning('*');
    }

    static getById(id, { trx, eagerOpts } = { eagerOpts: this.deepEagerOpts
}) {
        return this.query(trx)
            .findById(id)
            .eager(eagerOpts);
    }

    static getByChatUid(chatUid, { trx, sortBy, sortDir, skip, limit,
eagerOpts } = {
        sortBy: 'status',
        sortDir: 'asc',
        skip: 0,
        limit: Infinity,
        eagerOpts: this.deepEagerOpts
    }) {
        return this.query(trx)
            .where({ chatUid })
            .orderBy(sortBy, sortDir)
            .offset(skip)
            .limit(limit)
            .eager(eagerOpts);
    }
```

```
    static async countByChatUid(chatUid, { trx } = {}) {
        const result = await this.query(trx)
            .where({ chatUid })
            .count()
            .first();
        return result ? result.count : 0;
    }

    update(data, { trx } = {}) {
        return this.$query(trx)
            .patch(data)
            .returning('*');
    }

    getThing({ trx } = {}) {
        return this.$relatedQuery('thing', trx)
            .findOne({});
    }

    getChat({ trx } = {}) {
        return this.$relatedQuery('chat', trx)
            .findOne({});
    }

    toPublicJson() {
        let doc = pick(this, ['id', 'chatUid', 'thingId', 'status',
'pictureUrl', 'createdAt', 'updatedAt']);
        if (this.thing) {
            doc = {
                ...doc,
                ...pick(doc, ['name', 'price', 'lost'])
            };
        }
        return doc;
    }
};
const { Model } = require('objection');
const { pick } = require('lodash');

module.exports = class Things extends Model {
    static get tableName() {
        return 'things';
    }

    static get idColumn() {
        return 'id';
    }

    static get jsonSchema() {
        return {
            type: 'object',
            required: ['ownerUid', 'name'],
            properties: {
                id: { type: 'number' },
                ownerUid: { type: 'string' },
                name: { type: 'string' },
                price: { type: 'number' },
                lost: { type: 'boolean' },
                createdAt: { type: 'string' },
                updatedAt: { type: 'string' }
            }
        };
    }
```

```
    static get relationMappings() {
        return {
            owner: {
                relation: Model.BelongsToOneRelation,
                join: {
                    from: 'things.owner_uid',
                    to: 'users.uid',
                },
                modelClass: `${__dirname}/User`
            }
        };
    }

    static create(data, { trx } = {}) {
        return this.query(trx).insert(data).returning('*');
    }

    static getById(id, { trx } = {}) {
        return this.query(trx).findById(id);
    }

    static get({ trx, sortBy, sortDir, skip, limit } = {
        sortBy: 'name',
        sortDir: 'asc',
        skip: 0,
        limit: Infinity
    }) {
        return this.query(trx)
            .orderBy(sortBy, sortDir)
            .offset(skip)
            .limit(limit);
    }

    static getByOwner(ownerUid, { trx, sortBy, sortDir, skip, limit, lost }
= {
        sortBy: 'name',
        sortDir: 'asc',
        skip: 0,
        limit: Infinity,
        lost: null
    }) {
        const query = this.query(trx)
            .where({ ownerUid })
            .orderBy(sortBy, sortDir)
            .offset(skip)
            .limit(limit)
          .skipUndefined();
        if (lost === null) return query;
        else return query.orWhere({ lost });
    }

    static async countOwned(ownerUid, { trx } = {}) {
        const result = await this.query(trx)
            .where({ ownerUid })
            .count()
            .first();
        return result ? result.count : 0;
    }

    getOwner({ trx } = {}) {
        return this.$relatedQuery('owner', trx).findOne({});
    }

    update(data, { trx } = {}) {
```

```
        return this.$query(trx)
            .patch(data)
            .returning('*');
    }

    delete({ trx } = {}) {
        return this.$query(trx)
            .delete();
    }

    toPublicJson() {
        return pick(this, ['id', 'ownerUid', 'name', 'price', 'lost',
'createdAt', 'updatedAt']);
    }
};

const config = require('../config/server');
const App = require('./App');

const logger = require('log4js')
  .getLogger('server');

const app = new App(config);

function handleError(err) {
  logger.fatal('Unhandled exception occured %s', err);
}

async function terminate(signal) {
  try {
    await app.terminate();
  } finally {
    logger.info('App is terminated with signal: %s', signal);
    process.kill(process.pid, signal);
  }
}

// Handle uncaught errors
app.on('error', handleError);
// Start server
if (!module.parent) {
  const server = app.listen(config.app.port, () => {
    logger.info(`API server listening on ${config.app.port}, in
${config.env}`);
  });
  server.on('error', handleError);

  const errors = ['unhandledRejection', 'uncaughtException'];
  errors.forEach((error) => {
    process.on(error, handleError);
  });

  const signals = ['SIGTERM', 'SIGINT', 'SIGUSR2'];
  signals.forEach(signal => {
    process.once(signal, () => terminate(signal));
  });
}

module.exports = app;
const { Model } = require('objection');
const { pick } = require('lodash');
const uuid = require('uuid/v4');

class AuthData extends Model {
```

```
static get idColumn() {
  return 'uid';
}

static get tableName() {
  return 'auth_data';
}

static get relationMappings() {
  return {
    profile: {
      relation: Model.BelongsToOneRelation,
      modelClass: `${__dirname}/User`,
      join: {
        from: 'auth_data.uid',
        to: 'users.uid'
      }
    },
  };
}

static get jsonSchema() {
  return {
    type: 'object',
    required: [
      'email',
      'googleId',
    ],
    properties: {
      uid: { type: 'string' },
      refreshToken: { type: 'string' },
      googleId: { type: 'string' },
      email: { type: 'string' },
      createdAt: { type: 'string' },
      updatedAt: { type: 'string' }
    }
  };
}

async $beforeInsert(queryContext) {
  await super.$beforeInsert(queryContext);
}

async $beforeUpdate(queryContext) {
  await super.$beforeUpdate(queryContext);
}

static async isEmailTaken(email, { trx } = {}) {
  const result = await AuthData.query(trx)
    .where({ email })
    .count();
  return !result[0].count;
}

static findByEmail(email) {
  return AuthData.query()
    .findOne({ email });
}

static findByGoogleId(googleId, { trx } = {}) {
  return AuthData.query(trx)
    .findOne({ googleId });
}
```

Додаток З
Копія презентації

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО”

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ КОМП’ЮТЕРНИХ СИСТЕМ

Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS

Виконав: Казимиров Данило Миколайович

Керівник: доц. кафедри ПЗКС, к.т.н., доц. Заболотня Тетяна Миколаївна

Київ – 2020

АКТУАЛЬНІСТЬ

Щотижня в аеропортах США гублять 120 000 ноутбуків

За даними Нацполіції українцями за рік було повідомлено про втрату

12 000 документів

35 500 тис мобільних

АКТУАЛЬНІСТЬ

РЕЧІ, ЯКІ НАЙЧАСТІШЕ ГУБИЛИ У 2019 ЗА ДАНИМИ UBER

Можна знайти власника	Власник невідомий
ТЕЛЕФОН	КАМЕРА
ІД / ЛІЦЕНЗІЯ	ГАМАНЕЦЬ
	КЛЮЧІ
	РЮКЗАК
	ОДЯГ
	ОКУЛЯРИ
	НАВУШНИКИ
	VAPE / E-CIG

3/17

АНАЛОГИ

ПОСТАНОВКА ЗАДАЧІ

МЕТА ПРОЄКТУ:

Збільшити відсоток повернутих загублених речей їх власникам за допомогою розроблення **веб-додатку** для сприяння поверненню втрачених речей на основі платформи хмарних обчислень AWS

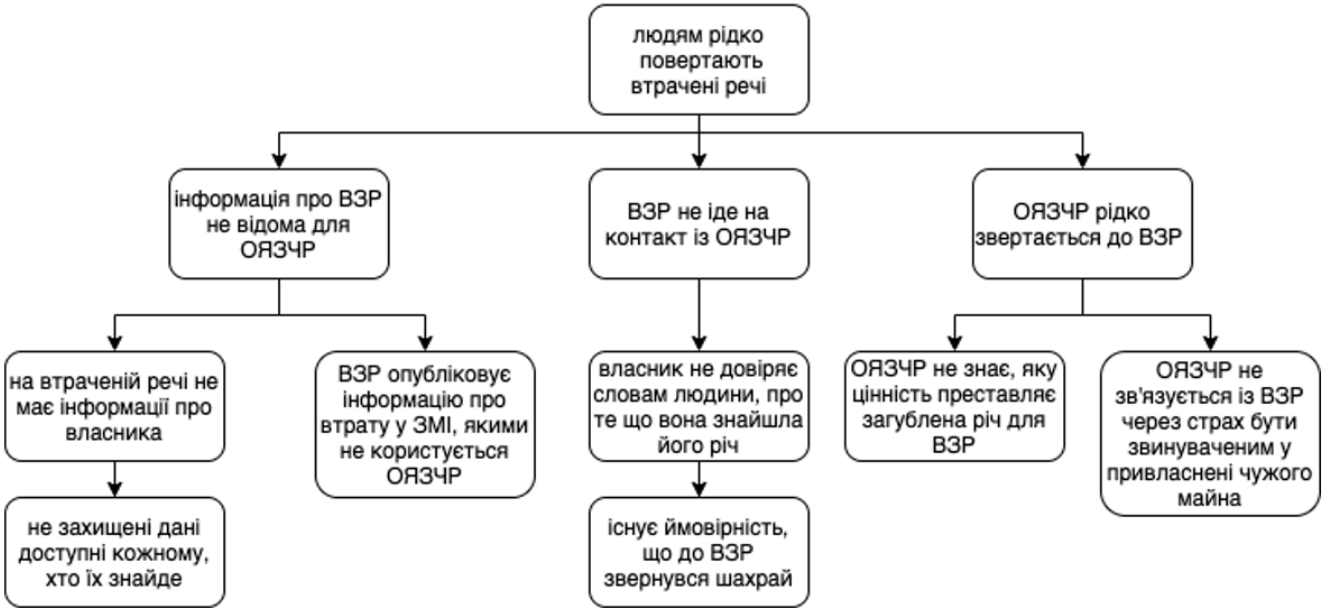
ПОСТАНОВКА ЗАДАЧІ

ЗАВДАННЯ:

1. Дослідити проблеми, які заважають людям отримувати загублені речі назад.
2. Проаналізувати існуючі програмні рішення.
3. Визначити вимоги до розроблюваного ПЗ.
4. Розробити веб-додаток у відповідності до сформованих вимог, який дозволить знаходити власника загубленої речі та встановлювати із ним контакт з метою повернути цю річ.
5. Провести тестування отриманого ПЗ та проаналізувати результати.
6. Знайти шляхи подальшого розвитку проєкту.

ДЕРЕВО ПРОБЛЕМ

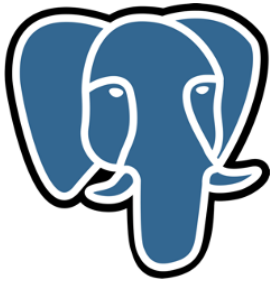
Дерево проблем

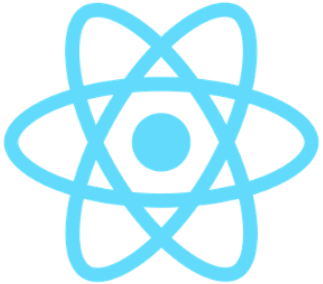


ОЯЗЧВ – особа, яка знайшла чужу річ

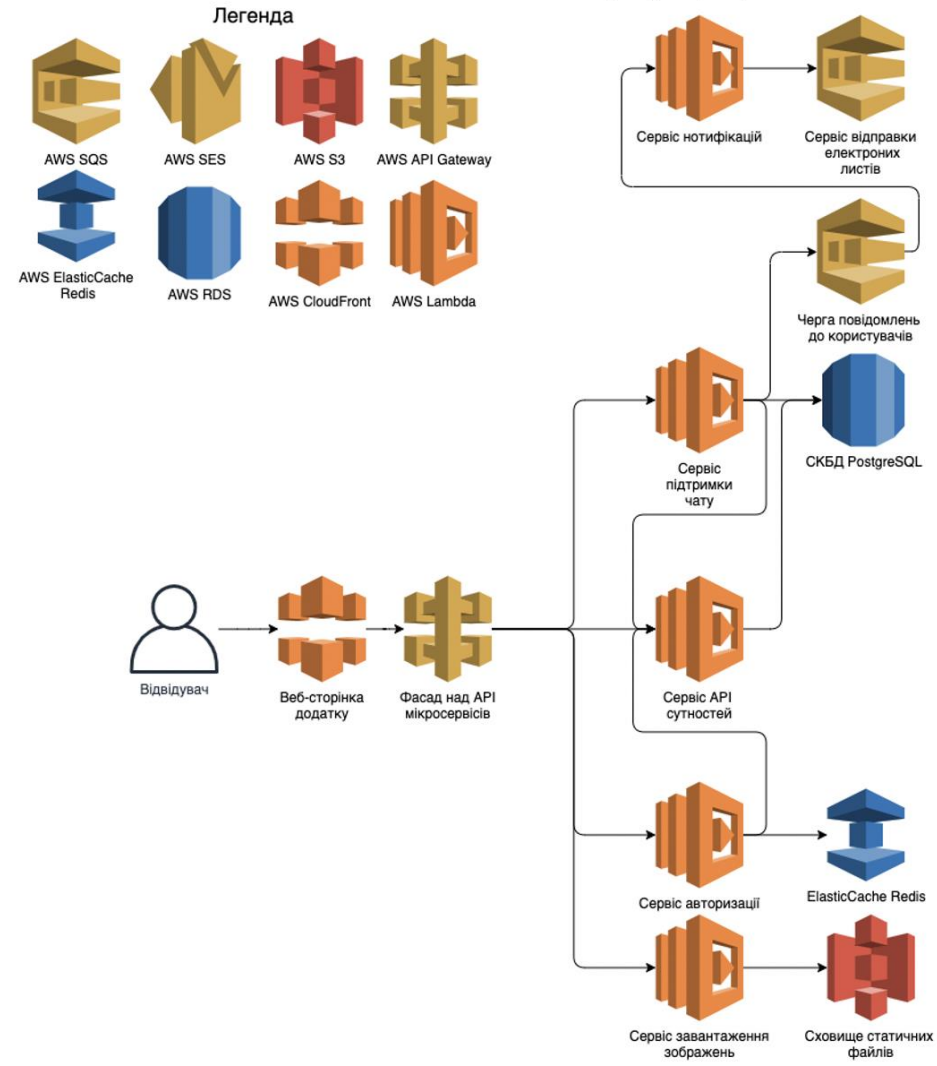
ВЗР – власник загубленої речі

ВИБІР ТЕХНОЛОГІЙ РОЗРОБЛЕННЯ

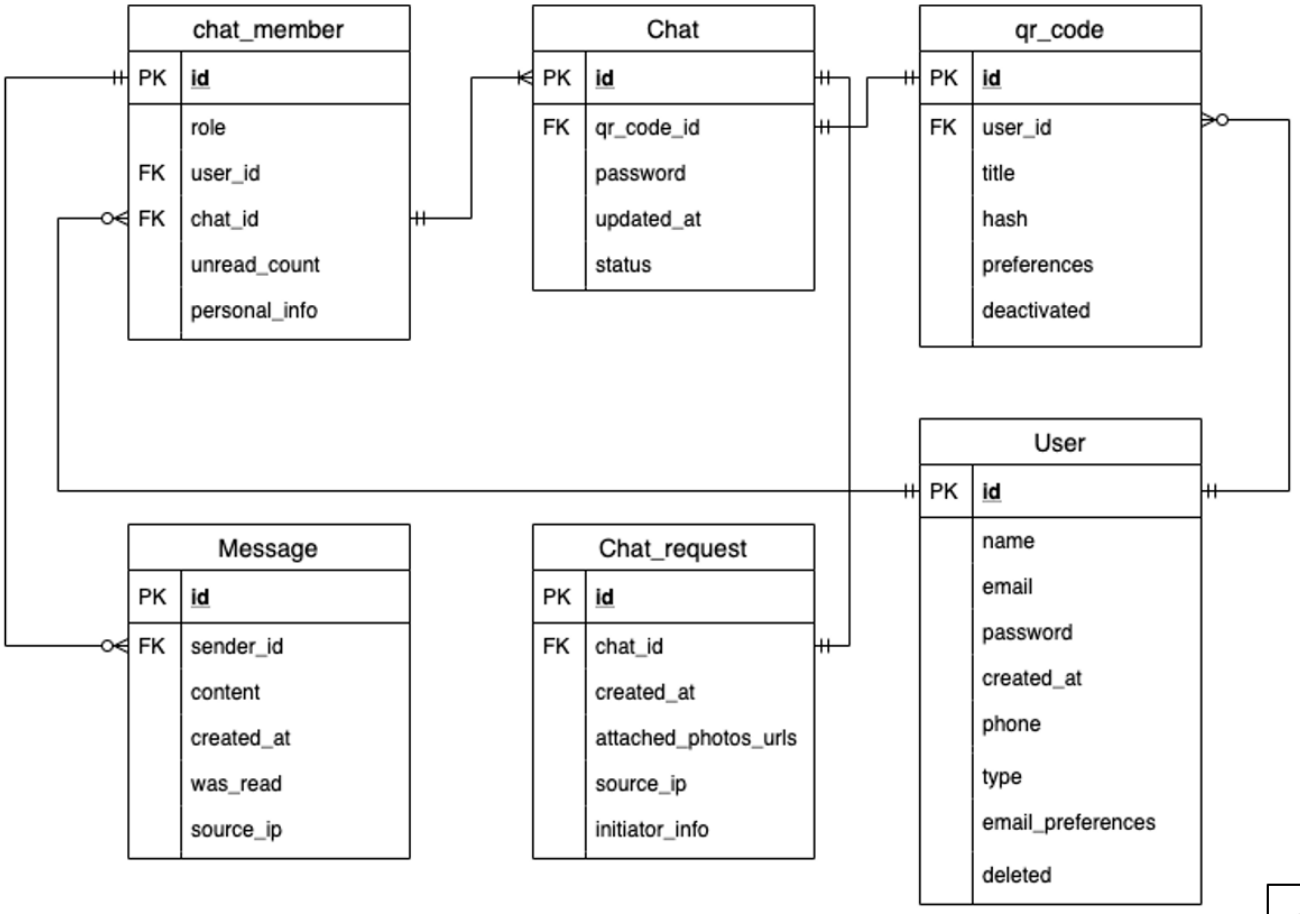




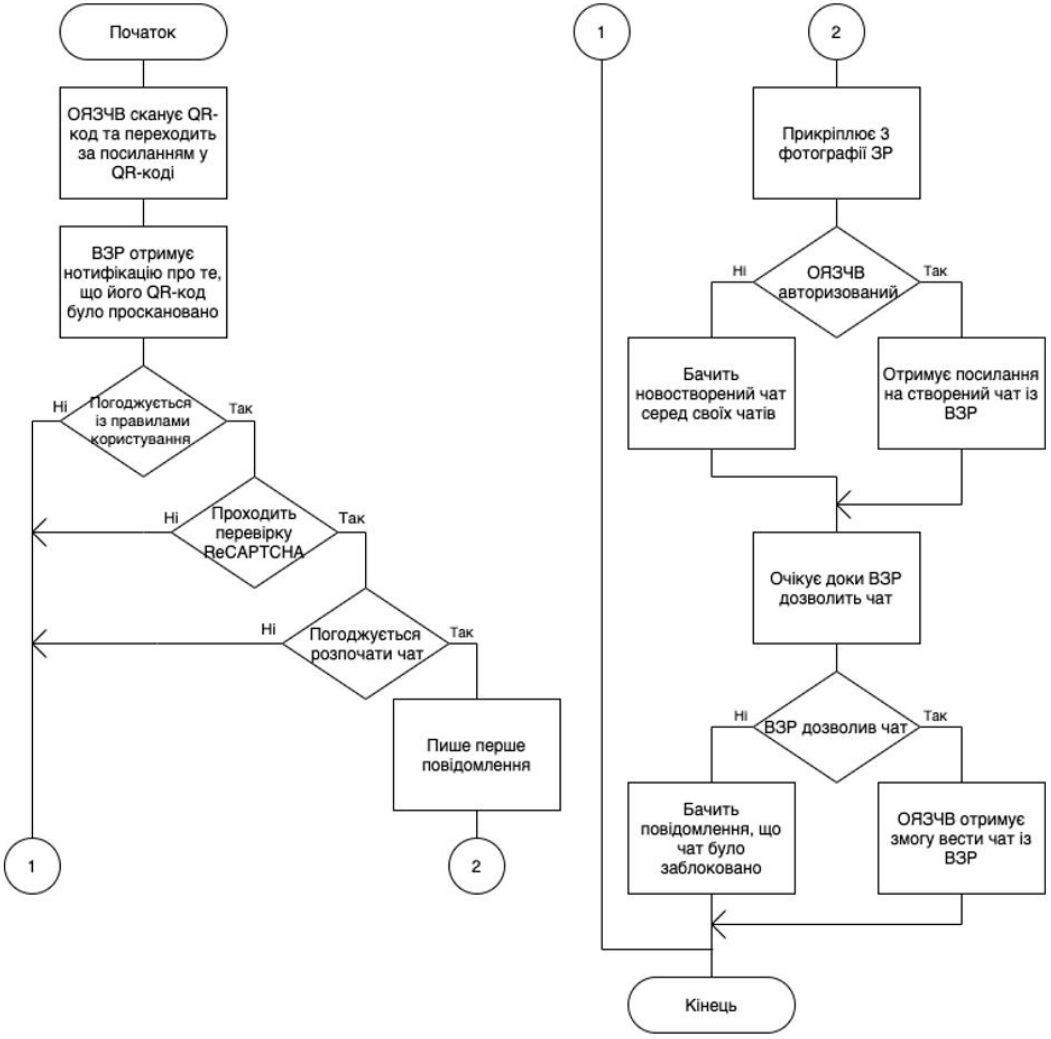
АРХІТЕКТУРА ВЕБ-ДОДАТКУ



СТРУКТУРА БД



ДІАГРАМА ДІЯЛЬНОСТІ. ВЗАЄМОДІЯ ОЯЗЧР ІЗ ВЕБ-ДОДАТКОМ



РОЗРАХУНОК ВАРТОСТІ ПЛАТФОРМИ AWS

Вартість за кожну тис. користувачів 25$

\+ вартість підтримки інфраструктури в цілому 70$

Сервіс	Використання	Вартість
AWS Lambda	2.5*10^6 GB/c	6.17$
AWS S3	1 GB / 100000 запитів	5$
AWS SES	50000 листів	5$
AWS SQS	5*10^6 запитів	2.2$
AWS APIGateway	1.5*10^6 запитів	1.8$
AWS RDS	1 x db.small	20$ (загалом)
AWS ElastiCache	1 x cache t2.medium	50$ (загалом)

ШЛЯХИ ПОДАЛЬШОГО РОЗВИТКУ ПРОЄКТ

- розширення функціональних можливостей з персоналізації QR коду: вибір кольору, форми, додавання власного логотипу тощо;
- інтеграція із сторонніми сервісами авторизації за протоколом OAtuh2;
- використання технології Push;
- отримання даних про геопозицію ОЯЗЧВ за його IP-адресою.

АПРОБАЦІЯ

1. Участь у V Міжнародній науково-практичній конференції “Інформаційні технології в освіті, науці й техніці” (ІТОНТ-2020). Результати опубліковані у вигляді тез доповіді.

2. Підготовлено пакет документів на отримання авторського свідоцтва.

ВИСНОВКИ

В процесі роботи над даним проєктом було зроблено:

1. Було досліджено проблеми, які заважають людям отримувати загублені речі назад.
2. Знайдено та проаналізовано існуючі програмні рішення.
3. Сформовано і визначено вимоги до розроблюваного ПЗ.
4. Розроблено веб-додаток, який дозволяє знаходити власника загубленої речі та встановлювати із ним контакт для того, аби повернути його власність.
5. Проведено тестування отриманого ПЗ та проаналізовано результати, на основі яких можна стверджувати, що розроблення виконано повністю та відповідає вимогам.
6. Знайдено шляхи подальшого розвитку проєкту.

ВИСНОВКИ

В рамках проєкту було **реалізовано**:

1. Веб-інтерфейс користувача.
2. Сервіс-орієнтовану архітектуру на основі платформи AWS.
3. Алгоритм генерації QR-кодів із захищеним посиланням на сторінку його власника.
4. Ведення чату між двома користувачами.
5. Алгоритм авторизації користувачів на основі технології JWT.
6. Процедуру завантаження зображень у хмарне сховище.
7. Процеси неперервної інтеграції та доставки коду.

Дякую за увагу!

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2019 р.

ВЕБ-ДОДАТОК ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS

Програма та методика тестування

ДП.045440-04-51

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Данило Казимиров

2019

ЗМІСТ

1. ОБ’ЄКТ ВИПРОБУВАНЬ

Веб-додаток для сприяння поверненню втрачених речей на основі платформи AWS створений на платформі Node.js в оточенні платформи AWS з використанням технології React.js та AJAX.

2. МЕТА ТЕСТУВАННЯ

У процесі тестування має бути перевірено наступне:

- функціональна працездатність елементів сторінок веб-додатку;
- наявність доступу до бази створених користувачем QR-кодів;
- можливість вести чат між користувачами;
- забезпечення належного рівня безпеки даних;
- зручність роботи з веб-сайтом;
- відповідність дизайну вимогам Технічного завдання.

3. МЕТОДИ ТЕСТУВАННЯ

Тестування виконується методом Gray Box Testing. Перевіряється як код, так і безпосередньо програмний продукт на відповідність функціональним вимогам. Тестування відбувається на рівні «системного тестування».

Використовуються наступні методи:

1) Функціональне тестування, зокрема на рівні інтеграційного тестування (тестування взаємодії різних модулів програми) та тестування інтерфейсу (тестування користувацької взаємодії).
2) Тестування продуктивності програмного забезпечення, зокрема Stability testing (тестування стабільності) та Load testing (навантажувальне тестування);
3) Тестування інтерфейсу.

4. ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування виконується засобами інструментарію SpecFlow.

Працездатність web-ресурсу перевіряється шляхом:

1) динамічного ручного тестування – введенням граничних та недопустимих значень в поля, які можна редагувати;
2) динамічного ручного тестування на відповідність функціональним вимогам;
3) тестування коду юніт тестами;
4) тестування веб-додатку в різних веб-браузерах;
5) тестування при максимальному навантаженні;
6) тестування стабільності роботи при різних умовах;
7) тестування зручності використання;
8) тестування інтерфейсу.

Факультет прикладної математики

Кафедра програмного забезпечення комп'ютерних систем

«ЗАТВЕРДЖЕНО»

Науковий керівник кафедри

_____ Іван ДИЧКА

«___» _____ 2020 р.

ВЕБ-ДОДАТОК ДЛЯ СПРИЯННЯ ПОВЕРНЕННЮ ВТРАЧЕНИХ РЕЧЕЙ НА ОСНОВІ ПЛАТФОРМИ AWS

Керівництво користувача

ДП.045440-05-34

«ПОГОДЖЕНО»

Керівник проєкту:

_____ Тетяна ЗАБОЛОТНЯ

Нормоконтроль:

_____ Микола ОНАЙ

Виконавець:

_____ Данило КАЗИМИРОВ

2020

ЗМІСТ

1. Опис структури вею-ресурсу

Веб-додаток для сприяння поверненню втрачених речей побудований за моделлю SPA, що зумовлює те, що більшість сторінок веб-додатку є динамічними та будуються в реальному часі на стороні клієнта за допомогою скриптів на мові програмування JavaScript та фреймворку React.js.

До статичних належать веб-сторінка «Головна» / «Index».

Динамічні сторінки веб-додатку представлені наступними веб-сторінками:

- персональна сторінка користувача;
- домашня сторінка користувача;
- сторінка редагування/видалення QR-коду;
- сторінка чату;
- сторінка ініціалізації чату між користувачем та ОЯЗЧВ;

Кожна веб-сторінка складається із таких блоків:

- шапка сайту;
- строчка навігації із шляхом до поточної сторінки;
- підвал сайту;

В цих компонентах знаходяться корисні посилання на сторінки, які будуть часто використовуватися, наприклад: домашня сторінка користувача, сторінка всіх його чатів, сторінка налаштувань профілю.

2. Опис вмісту статичних веб-сторінок

Головна сторінка веб-додатку виступає рекламним банером, та надає відвідувачам інформацію про призначення та головну ідею цього веб-сайту. На ній розміщено наступні кнопки: «Зареєструватися» та «Увійти». При натисканні на кнопку «Зареєструватися», з’являється спливаюче вікно реєстрації. При натисканні на кнопку «Увійти», з’являється спливаюче вікно авторизації.

Задній фон сторінки представляє собою анімоване зображення, на якому відтворено ефект старої плівки, для того аби фон контрастував із текстом на ній.

Якщо користувач авторизований, за посиланням на головну сторінку веб-додатку буде відкриватися домашня сторінка користувача. Це зроблено для того, аби не заплутивати користувача, відображаючи йому кнопки авторизації та інформацію про веб-додаток після того, як він увійшов у свій акаунт.

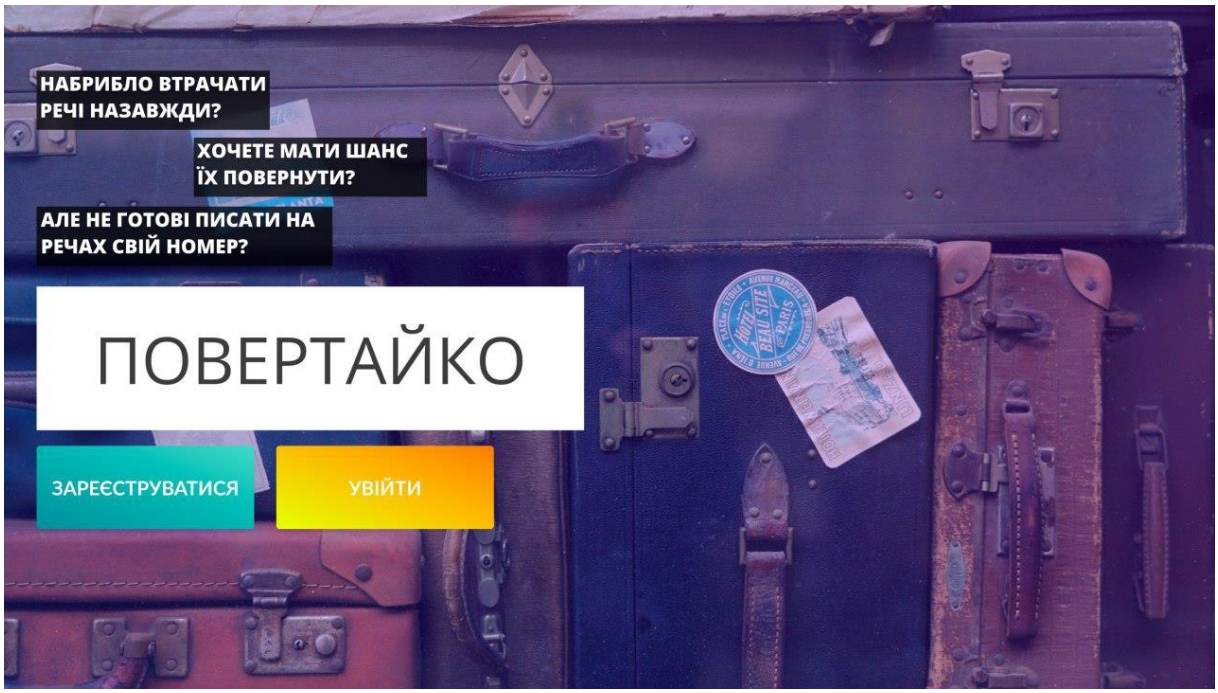


Рис. 1. Головна сторінка веб-додатку

3. Процедура авторизації користувача

Користувачі веб-додатку можуть авторизуватися не покидаючи поточної сторінки. Це реалізовано за допомогою вспливаючих вікон. Ідея їх використання полягає в тому, що на поточній сторінці, після активації, з'являється компонент, який перекриває собою всю сторінку та акцентує увагу відвідувача на свій вместі. В такому вікні було розміщено форму авторизації та реєстрації користувачів.

Для того, аби програма могла примусово відсилати користувача на сторінку авторизації, було реалізовано окрему сторінку, на якій одразу відображається вікно логіну (рис. 2). Після успішного заповнення форми, користувач буде пересланий на сторінку, посилання на яку було передано в параметрах поточної URL.

Форма авторизації складається із таких елементів: поле вводу адреси електронної пошти та поля вводу паролю.

Для забезпечення безпеки даних відвідувача, пароль, який він вводить буде заховано за зірочками. Дане вікно містить посилання на форму відновлення паролю.

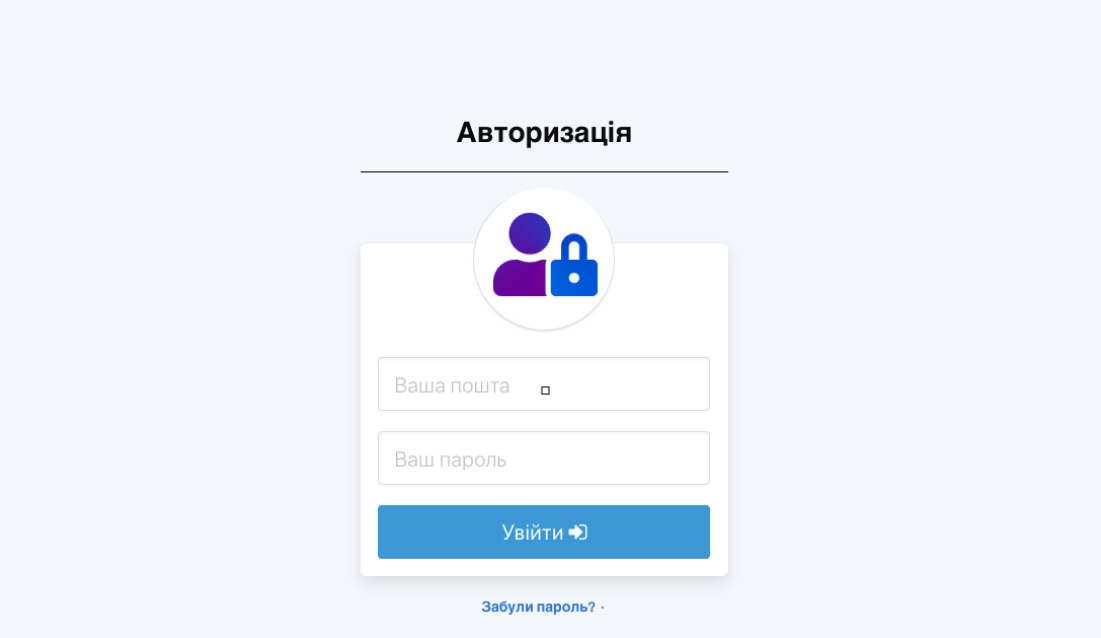


Рис. 2. Спливаюче вікно із формою для авторизації користувача

4. Домашня сторінка користувача

Домашня сторінка користувача представляє собою доску, на якій зібрано усю важну для користувача інформацію, а саме:

- список QR-кодів, із якими взаємодіяли нещодавно;
- список останніх отриманих повідомлень у кожному чаті;

В блоці зі списком QR-кодів, із якими нещодавно взаємодіяли, користувач може побачити короткий список із 10 QR- кодів, які він редагував, чи за якими отримував повідомлення (рис. 3).

На домашній сторінці користувач розміщенні всі QR-коди користувача та поштову скриньку із останніми отриманими повідомленнями від ОЯЗЧВ про знайдені речі (рис. 3).

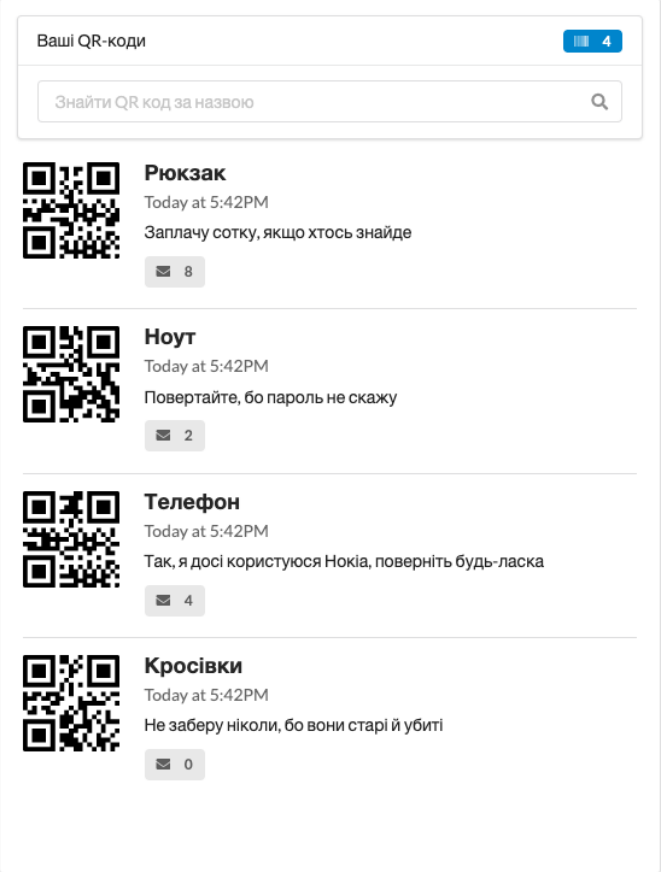


Рис. 3. Блок списку QR-кодів, які було оновлено нещодавно на домашній сторінці користувача

Блок із останніми отриманими повідомленнями відображає користувачу всі повідомлення, які він отримав нещодавно та не встиг прочитати (див. рис. 4). При цьому відображаються лише по одному повідомленню із кожного чату. Якщо чат було деактивовано, повідомлення виводитися не буде. Якщо користувач натисне на повідомлення зі списку, воно стане обраним та буде підсвічуватися, стануть доступні дії, які знаходять зверху над цим блоком. Ці дії включають в себе: прочитати повідомлення, заблокувати чат, перейти до чату. При обранні користувачем останнього варіанту – він перейде на сторінку чату, до якого належить обране повідомлення.

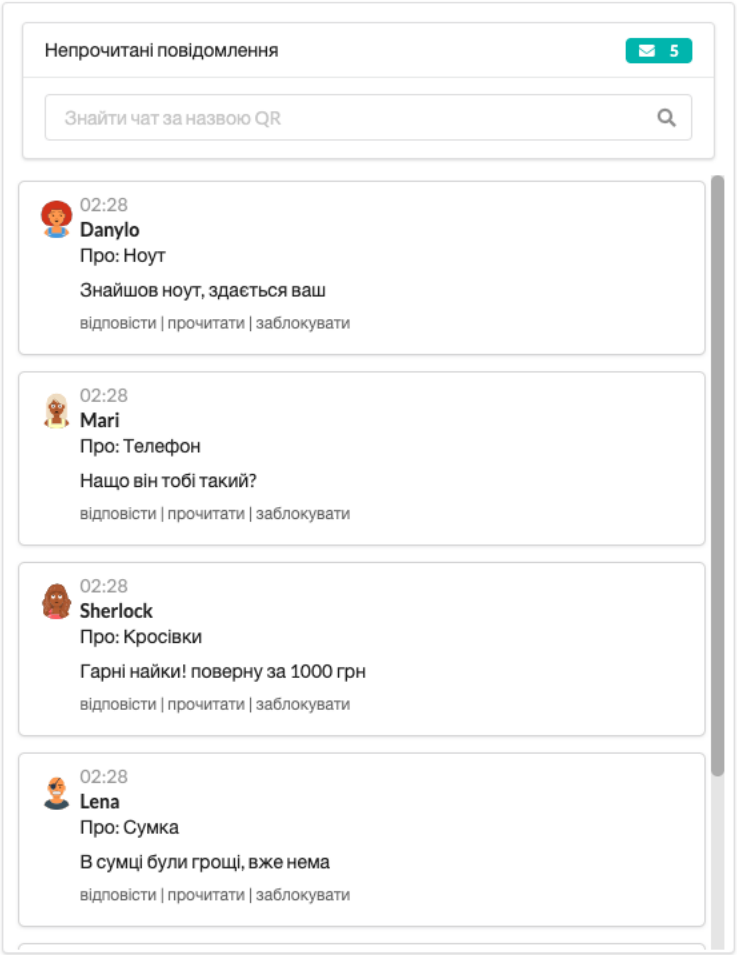


Рис. 4. Блок із списом останніх отриманих непрочитаних повідомлень

5. Процедура створення нового QR-коду

Функціональними вимогами до веб-додатку для сприяння поверненню втрачених речей передбачається реалізація створення користувачем нового QR-коду, який він може завантажити на локальний комп’ютер та розмістити на своїй власності (рис. 5).

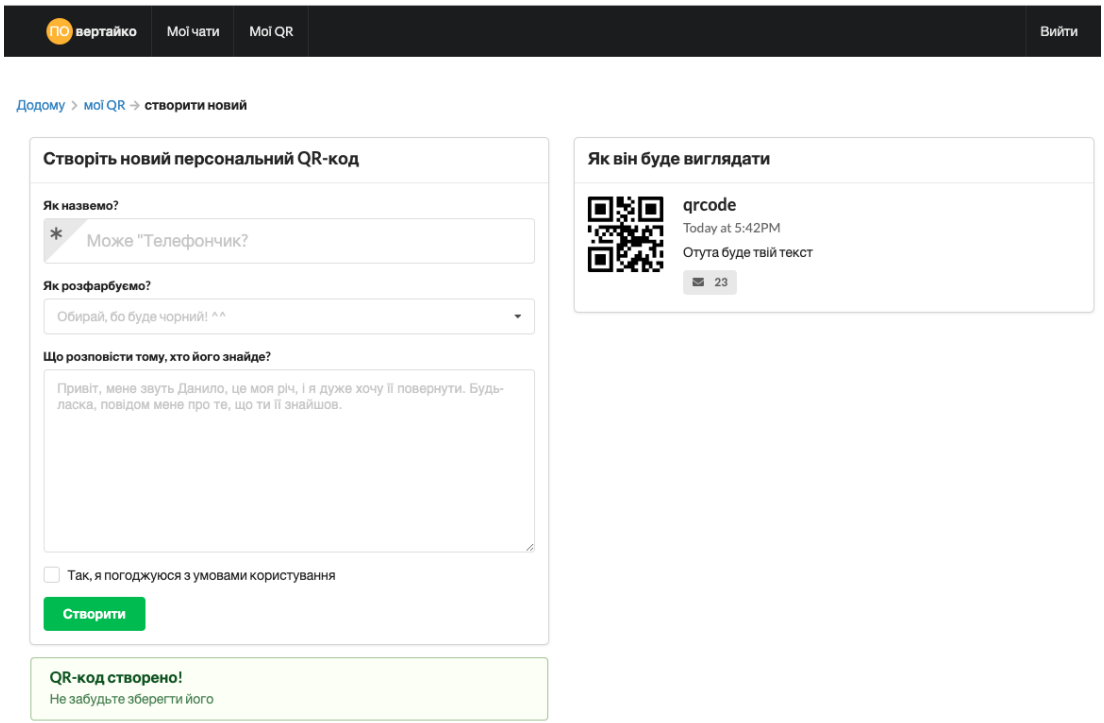


Рис. 5. Сторінка створення нового QR-коду користувачем

Ця сторінка вміщує форму, в якій розміщено поля, куди користувач може вводити інформацію про QR-код. А саме:

- давати QR-коду назву;
- писати повідомлення, яке побачить ОЯЗЧВ коли відсканує QR-код та перейде за посиланням;
- обирати стиль QR-коду;

Обабіч форми розміщено зображення QR-коду яке користувач, зручним йому способом повинен розмістити на своїй власності. Під цим зображенням знаходиться кнопка для завантаження зображення QR-коду у високому розширенні на локальний комп’ютер.

6. Сторінка чату між ВЗР та ОЯЗЧВ

Для того, аби користувачі веб-додатку могли вести бесіду між собою, було реалізовано сторінку чату. На цій сторінці відображається зона із скролом, в якій розміщено повідомлення від ВЗР та ОЯЗЧВ (рис. 6).

Кожне повідомлення представляє собою прямокутник, в якому відображається інформація про відправника: ім’я, згенерований аватар користувача, локація, звідки було відправлено повідомлення (якщо така інформація відома системі), час відправки повідомлення та його текст.

Три перших зображення, які ОЯЗЧВ повинен надати ВЗР для ініціації чату також відображаються в зоні із повідомленнями.

Для того аби відправити повідомлення, користувач має написати його текст у текстовому контролері та натиснути кнопку «надіслати».

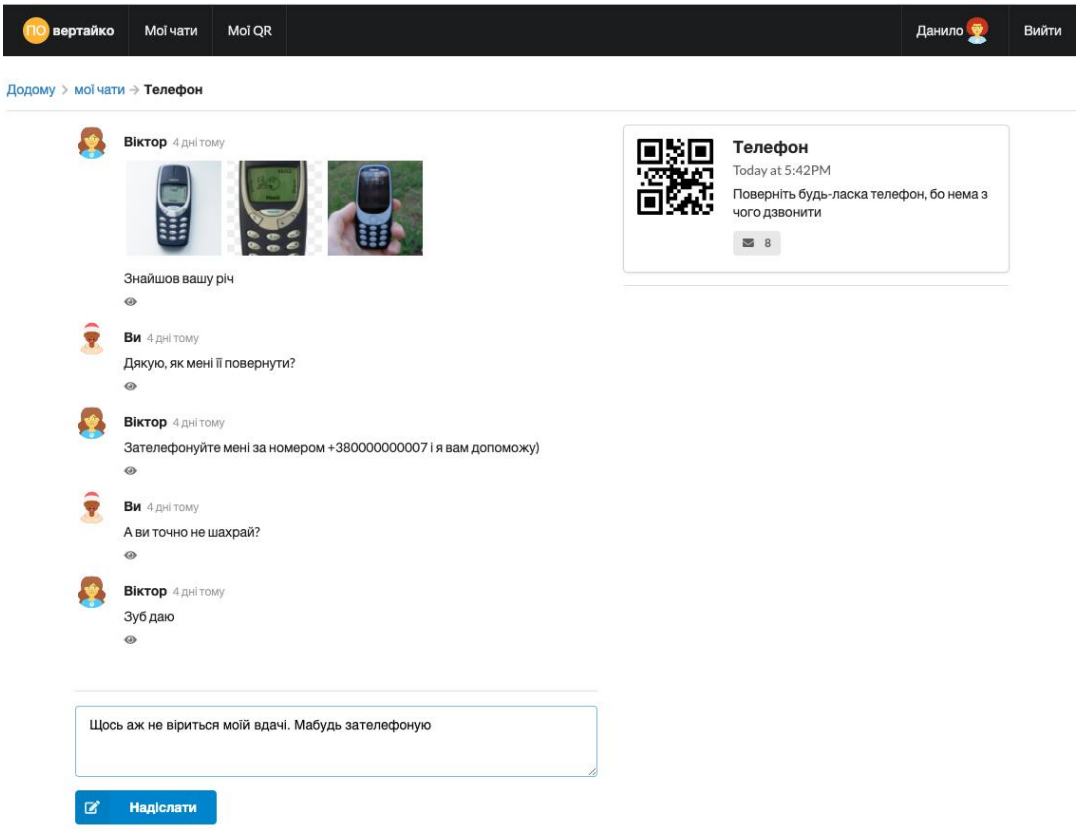


Рис. 6. Сторінка чату між ВЗР та ОЯЗЧВ